UNIVERSITETET I OSLO
Institutt for informatikk

En referansemodell for kosteffektive datafangstløsninger basert på tjenesteorienterte prinsipper

Masteroppgave

Anne-Marte Krogsrud

1. mai 2006

Forord

Denne oppgaven er skrevet som en del av min mastergrad ved Institutt for Informatikk, Universitetet i Oslo.

Studietiden ved Universitetet i Oslo har vært spennende, utfordrende og lærerrik. Ikke minst har jeg opplevd masterstudiet de to siste årene som spesielt interessant. Jeg har mange å takke for at studietiden ved universitetet er en periode jeg er stolt av og vil se tilbake på med glede, og ikke minst har jeg mange å takke med tanke på arbeidet med masteroppgaven som nå avsluttes.

Først vil jeg få takke Statistisk sentralbyrå, og spesielt Hanne Mette Janson, som gjorde denne masteroppgaven mulig. Jeg må også få rette en stor takk til Jon Folkedal som, sammen med Rannveig Aasen og Lars Pedersen, har vært til stor hjelp gjennom samtaler og diskusjoner underveis.

Jeg vil selvfølgelig også få takke min veileder ved Universitetet i Oslo, Aurelie Aurilla Arntzen, som med sitt engasjement og kritiske røst har gitt meg gode og konstruktive innspill, som igjen har ført til økt motivasjon og arbeidslyst.

En stor takk vil jeg også få rette til mine medstudenter og studiner som har vært viktige motivatorer for meg og som har deltatt i gode diskusjoner omkring mine problemstillinger.

Til slutt vil jeg takke min familie, og spesielt Trond, som har vært en uvurderlig støtte for meg gjennom dette arbeidet, og som med kritiske spørsmål og interesse for masteroppgaven har bidratt til progresjon i arbeidet, i tillegg til å ha fungert som korrekturleser for det ferdige produktet.

Oslo, 1. mai 2006

Anne-Marte Krogsrud

Innhold

Figurer

Tabeller

Abstrakt

Statistisk sentralbyrå (SSB) har, i følge Statistikkloven, rollen som hovedleverandør av offentlig norsk statistikk, og har dermed en kritisk virksomhet knyttet til innsamling av sitt datagrunnlag, kalt datafangst. IT-miljøet i SSB har som ambisjon å ta en ledende rolle innen elektronisk datafangst i sitt felleskap av offentlige europeiske statistikkbyråer. IT-avdelingen i SSB er derfor organisert rundt kjerneprosessene dataregistrering, kontroll og revisjon, metodisk utforming av spørreskjema, samt utvikling og drift av infrastruktur for elektroniske innsamlings- og mottakssystemer.

For å kunne nå ulike brukergrupper (avgivere av data) har SSB rettet sitt utviklingsarbeid rundt tre datafangstløsninger; Kommune-stat rapportering (Kostra), Informasjon- og datautveksling med næringelivet (Idun) og Altinn. Kostra og Idun er utviklet internt i SSB, mens Altinn er et samarbeidsprosjekt mellom Skattedirektoratet, Brønnøysundregistrene og SSB. SSB har undertiden sett at det er lite kostandseffektivt å inneha produktansvar for tre så like systemer.

Denne masteroppgaven består derfor av to deler; utviklingen av en fremtidig referansemodell for en integrasjon av Kostra og Idun gjennom bruk av en tjenesteorientert arkitektur, og utviklingen av en generisk nytte-/kostmodell for å evaluere IT-utviklingsprosjekter i offentlig sektor.

Dokumentet gir derfor en introduksjon til tjenesteorienterte arkitekturer og konseptene og historien bak denne måten å designe IT-systemer på, og teknologier brukt for å realisere en slik arkitektur. Spesielt blir Web services presentert som et mulig teknologivalg. Deretter følger en diskusjon av ulike former for systemintegrasjon og jeg presenterer en arkitekturmodell for en integrasjon av Kostra og Idun, hvor jeg samtidig beskriver enkelte krav til et slikt system. Til slutt gir jeg en innføring i evalueringer av IT-prosjekter. Jeg ser spesielt på nytte-/kostanalyser, og viktigheten av slike evalueringer, før jeg presenterer et forslag til en nytte-/kostmodell og beskriver hvordan denne kan implementeres i offentlig sektor.

Organisering av dokumentet

Da problemstillingen for oppgaven er todelt med systemintegrasjon på den ene side og evaluering av IT-utviklingsprosjekter på den andre siden har også dokumentet en tilsvarende inndeling.

Den delen av dokumentet som beskriver tjenesteorientert arkitektur som utviklingsprinsipp og systemintegrasjonen av Kostra og Idun er

inneholdt i Kapittel 2 og Kapittel 3. Kapittel 4 inneholder del to av problemstillingen, og omhandler evaluering av IT-prosjekter. Under følger en nærmere beskrivelse av hva de ulike kapitlene i dokumentet inneholder.

Kapittel 1 - Kontekst for studiet

Dette kapittelet gir en overordnet introduksjon til masteroppgaven og beskriver studiets kontekst som i hovesak er SSB, Kostra og Idun. Kapittelet gir også en generell introduksjon til datasystemer, systemintegrasjon og evaluering av IT-prosjekter.

Kapittel 2 - Systemintegrasjon, state of the art

Kapittel 2 gir en teoretisk innføring i prinsippene bak tjenesteorientert arkitektur. Kapittelet gir en oversikt over, og en kort sammenlikning av, CORBA, IBM Web Sphere, SAP NetWeaver og Web services som alle er teknologier som brukes for å realisere tjenesteorienterte arkitekturer. Kapittelet gir dernest en oversikt over de viktigste mekanismene og byggestenene i en tjenesteorientert arkitektur realisert med Web services.

Kapittel 3 - Arkitekturmodell

Kapittel 3 beskriver IT-arkitektur generelt og ulike syn på dette, også i lys av tjenesteorienterte arkitekturer i offentlig sektor. Kapittelet gir en nærmere innføring i Kostra og Idun slik disse systemene er representert i dag, og beskriver dernest en mulig referansearkitektur for en integrert systemløsning av Kostra og Idun.

Kapittel 4 - Evaluering av IT-utviklingsprosjekter

Kapittel 4 tar for seg evaluering av IT-utviklingsprosjekter, og gir først en generell introduksjon til systemutvikling i et historisk perspektiv. Kapittelet presenterer deretter ulike evalueringsteknikker, som ex-ante og ex-post evalueringer, og presenterer til slutt en nytte-/kostmodell til bruk på IT-utviklingsprosjekter i SSB.

Kapittel 5 - Konklusjon

Konklusjonen i Kapittel 5 gir en oppsummering av masteroppgaven og dette dokumentet som helhet, samt beskriver hva som kreves for å gjennomføre en reell integrasjon av Kostra og Idun, og for å gjøre en god nytte-/kostanalyse av en slik løsning.

Tillegg A

Tillegg B inneholder alle modellene som hører med under designet av referansearkitekturen for Kostra og Idun i Kapittel 3.

Kapittel 1

Kontekst for studiet

Informasjons- og kommunikasjonsteknologi (IKT) er etter hvert blitt en del av alles hverdag, og de fleste er i kontakt med slik teknologi enten privat eller gjennom arbeidet. I arbeidssammenheng benytter organisasjoner seg gjerne av IKT for å bedre dataflyten i organisasjonen, og for å oppnå bedre informasjonsflyt mellom personer og avdelinger gjennom bruk av blant annet e-post og meldingsutveksling.

Statistisk sentralbyrå (SSB) kan på mange måter sies å stå i en særstilling med tanke på data- og informasjonsflyt. SSB er det sentrale norske organet for innsamling, bearbeiding og formidling av offentlig statistikk (Statistisk sentralbyrå 1999), og det er dermed gitt at SSB henter og samler inn data fra et utall ulike datakilder, det være seg privatpersoner, offentlige kontorer eller private bedrifter i norsk næringsliv. SSB leverer også data til en rekke ulike mottakere av statistikk.

SSB har innsett at det på sikt er lite lønnsomt å ha produktansvar for både Kostra, Idun og AltInn og de ønsker dermed å etablere en fremtidig referansemodell, i første rekke for Kostra og Idun, i lys av tjenesteorienterte arkitekturer og å etablere en nytte-/kostmodell for gevinstrealisering av IT-utviklingsprosjekter i SSB.

1.1 Datasystemer

Én enkelt organisasjon administrerer gjerne mange ulike datasystemer. Antallet formater som data lagres på er nesten like høyt som antallet informasjonslagre, og for at en applikasjon skal kunne håndtere de ulike informasjonskildene i tillegg til grensesnittene brukt for å eksponere informasjonen, må man skrive kode spesielt til hver datakilde (Patrick 2005). At data i forskjellige organisasjoner er ulikt representert er isolert sett ikke et stort problem for organisasjonene, men ser man for eksempel dette i sammenheng med innrapportering av data til SSB får dette konsekvenser også utenfor organisasjonen. SSB kan ikke uten videre gå ut fra at data fra et datasystem i en organisasjon er på et gitt format, og rapporteringen blir dermed i større grad manuell i stedet for at denne kunne vært automatisert med direkte filuttrekk fra organisasjonenes ulike fagsystemer.

Når man snakker om integrasjon av gamle[1] og nye system i en organisasjon er det viktig å ha en holistisk tilnærming for på den måten å sørge for at flere ulike perspektiv som forretningsprosesser, IT-strategi, forretningsstrategi, mål og planer for organisasjonen tas i betraktning (Chen & Doumeingts 2003). Fokuset på en holistisk tankegang blir stadig sterkere fordi det er viktig for en effektiv organisasjon å ha IT-systemer som støtter opp om organisasjonens strategier og forretningsprosesser.

Organisasjoner har ofte flere ulike systemer til internt bruk. Mange organisasjoner har egne IT-avdelinger og utvikler systemer de trenger til internt bruk "på huset". Fordelene med dette er at systemene blir spesialtilpassede organisasjonens behov, samt at de gjerne utvikles av ansatte som er i relativt nær tilknytning til områdene de skal benyttes i, og som dermed har kjennskap til systemenes anvendelsesområde. Ulempene ved slike systemer kommer gjerne til syne når systemene trenger vedlikehold og oppgradering. Slike systemer følger ikke alltid en standard utviklingsmetodikk, de er ofte utviklet ad-hoc i forhold til ønsker og behov, noe som også gjerne har innvirkning på i hvilken grad systemene er dokumentert. Eierskapet til slike systemer kan bli uforholdsmessig stort for enkelte utviklere, som ikke ønsker å forkaste systemer de selv har brukt ressurser på å utvikle. Samtidig er kunnskapsdelingen i slike prosesser alltid en utfordring både av tidsmessige årsaker, og fordi ansatte ikke ønsker å dele sin kunnskap med andre ansatte av frykt for selv å bli overflødige.

Innkjøpte systemer har den fordelen at de gjerne er utviklet med standardiserte verktøy og metoder, og dermed benytter standardiserte dataformat. I tillegg skjer mye av vedlikeholdet av systemene gjennom oppdateringer og nye versjoner fra leverandøren, slik at behovet for vedlikeholdsarbeid for den lokale IT-avdelingen blir mindre. Ulempene er blant annet høye utgifter til produktlisenser, samt at det kan være vanskelig å tilpasse et gitt system til organisasjonen, både med tanke på eksisterende forretningsprosesser, arbeidsprosesser og til eksisterende systemer. Mellom innkjøpte systemer og internt utviklede systemer finner vi open source systemene. Slike systemer har de senere årene blitt mer og mer vanlige, og man kan på mange måter si at disse systemene blir for en mellomting å regne sett i forhold til de to ovennevnte systemkategoriene. Open source systemer leveres, som navnet tilsier, med åpen kildekode, slik at alle som benytter seg av disse står fritt til å endre kildekoden og tilpasse systemene etter eget behov. Dette krever imidlertid høy kompetanse hos utviklerne i organisasjonen fordi man må sette seg inn i programkode skrevet av andre, og man må også være forsiktig slik at man ikke endrer noen av funksjonene i systemet samtidig som man ødelegger systemet i forhold til funksjoner man ønsker å bevare slik de er.

Det er ofte en stor utfordring å integrere systemer og å få ulike systemer til å samarbeide og til å "snakke sammen". Dette gjelder såvel integrasjon av gamle systemer som nye systemer, og kombinasjoner av disse. Utfordringene kommer blant annet av at ulike systemer gjerne

[1] fra det engelske uttrykket "legacy-systems"

er utviklet med ulike programmeringsspråk og dermed også ofte med ulike abstraksjonsnivå i forhold til maskinens hardware. Samtidig kan systemene benytte ulike datatyper for samme data, noe som igjen kan føre til problemer med konvertering av data fra ett system til et annet.

Som jeg var inne på tidligere vil også grensesnittene til de ulike systemene variere. Samhandling mellom systemer er likevel viktig da dette ikke bare er med på å koble sammen avdelinger i en organisasjon, men også å øke samhandlingen mellom ulike organisasjoner, slik at man kan få et nettverk av systemer som utveksler data og informasjon imellom seg. Dette krever imidlertid god planlegging av systemarkitektur, grensesnitt mellom systemene, begrunnede valg i forhold til gjenbruk av eksisterende komponenter, man må ta hensyn til forretningslogikken for de ulike virksomhetene og sist men ikke minst må man sørge for at systemet støtter opp om forretningsprosessene i virksomhetene som skal bruke systemet.

Moderniseringsdepartementet (nå Fornyings- og administrasjonsdepartementet) har nedsatt et organ, "Kontrollorganet for eForvaltning". Dette organet skal på overordnet nivå skal sørge for elektronisk samhandling i og med offentlig sektor for å bidra til flere og bedre brukerrettede tjenester til næringsutvikling og i forhold til bedre bruk av offentlige ressurser. Ser man dette i sammenheng med målene som trekkes opp i ENorge 2009 er det på høy tid å prioritere eGovernment i en offentlig institusjon som SSB (Moderniseringsdepartementet 2005).

1.2 SSB som en IT-organisasjon

SSB arbeider for å dekke nasjonale og internasjonale behov for offentlig norsk statistikk, og analyser basert på denne. Den øvre ledelsen i SSB består av et styre og en administrerende direktør, mens den interne organiseringen er gjort i form av avdelinger, med underliggende seksjoner. IT-miljøet i SSB har som ambisjon å ta en ledende rolle innenfor elektronisk datafangst i sitt felleskap av offentlige europeiske statistikkbyråer. Som en konsekvens av dette har SSB organisert sin IT-avdeling til å inkludere alle kjerneprosesser innenfor datafangstløpet, herunder dataregistrering, kontroll- og revisjon, metodisk utforming av spørreskjema, og utvikling/drift av infrastruktur for elektroniske innsamlings- og mottakssystemer. Seksjonen hvor jeg skal foreta min analyse, seksjon for IT-utvikling, er underlagt avdeling for IT og datafangst. Seksjon for IT-utvikling har i øyeblikket 19 årsverk, og er knyttet til utvikling av løsningene nevnt over.

Avdeling for IT og datafangst har både ansvar og myndighet i IT-faglige spørsmål, blant annet slik at IT-utvikling i organisasjonen skal integreres og tilpasses de rammer og retningslinjer som trekkes opp for SSB. Seksjonens ansatte er til enhver tid engasjert i flere ulike prosjekter for å utvikle og vedlikeholde programvareløsninger og systemarkitektur både til internt bruk innad i SSB, og IT-løsninger mot eksterne brukere til bruk ved elektronisk innrapportering.

1.2.1 Rammer for virksomheten

Det er lover, internasjonale krav, tilgang på kompetanse og finansiering som i grove trekk setter rammene for SSBs virksomhet. SSB er administrativt sett underlagt Finansdepartementet og har et regjeringsoppnevnt styre (Statistisk sentralbyrå 1999). Statistikkloven (av 16. juni 1989 nr. 54) setter de formelle rammene for all norsk offisiell statistikk (Statistisk sentralbyrå 2002), men loven er generell nettopp for å fremme effektiv produksjon av statistikk gjennom å gi regler for innsamling og bruk av opplysninger for statistiske formål. I Statistikkloven beskrives SSBs ansvar som blant annet å samordne omfattende statistikk som utarbeides av forvaltningsorganer. Statistikkloven pålegger SSB restriktiv bruk av innsamlede data, slik at for eksempel individdata om enkeltpersoner eller bedrifter skal behandles fortrolig og at det aldri skal publiseres opplysninger om disse. Personopplysningsloven gir sammen med Datatilsynet retningslinjer for hvordan individopplysninger skal behandles, og det gjelder særlig strenge rutiner for sensitive personopplysninger.

Ifølge Statistikkloven har SSB ansvar for å kartlegge og prioritere behov for offentlig statistikk, samordne statistikk som utarbeides av ulike forvaltningsorganer, utvikle statistiske metoder og utnytte statistikken til analyse og forskning, gi opplysninger til statistisk bruk for forskningsformål og for offentlig planlegging. I tillegg har SSB, fra norsk side hovedansvaret for Norges internasjonale statistiske samarbeid. Statistikkloven sier videre at SSB er en faglig uavhengig institusjon, noe som betyr at SSB selv er ansvarlige for faglig innhold i statistikk og analyser, at de selv bestemmer hva de skal publisere av offentlig statistikk, samt når og hvordan dette skal gjøres (Statistisk sentralbyrå 1999).

Datasikkerhet og personvern er en avgjørende forutsetning for tillit hos de mange oppgavegiverne, og er en sentral del av SSBs verdigrunnlag som kan oppsummeres i brukerorientering, integritet, effektivitet og fornyelse (Statistisk sentralbyrå 2004). Samfunnsutviklingen må avspeiles i statistikken. Hovedprioriteringer i statistikken er derfor struktur og utvikling i norsk økonomi, befolkning og levekår, miljøstatistikk, tjenesteytende virksomhet, offentlig sektors ressursbruk, næringslivet, egen forskningsvirksomhet samt internasjonal statistisk sammenlikning og standardisering. Markedsintegrering og globalisering øker graden av samarbeid om internasjonale statistiske standarder, og statistikksamarbeidet med EØS påvirker mer enn 50 prosent av statistikkproduksjonen (Statistisk sentralbyrå 2004).

I og med at SSB er underlagt Finansdepartementet er deres virksomhet først og fremst finansiert av grunnbevilgninger over statsbudsjettet, kalt statsoppdraget (Statistisk sentralbyrå n.d.*b*). Utover denne finansieringen får SSB også inntekter ved at offentlige og private oppdragsgivere betaler direkte for utvikling og tilrettelegging av statistikk og analyser, kalt brukerfinansierte oppdrag.

1.2.2 Informasjonssamfunnet

Brukerbehovene for statistikk endres i takt med endringene i samfunnet, og kunnskap og informasjon betyr stadig mer både i næringslivet og i offentlig virksomhet. Overgangen til hva SSB kaller informasjonssamfunnet har ført til en kraftig økning av informasjonsflommen i samfunnet, og den offisielle statistikken har dermed en stadig viktigere rolle i form av å omdanne denne informasjonen til kunnskap for å gi innsikt i nettopp samfunnsendringene. Samfunnsutviklingen gir også nye utfordringer og muligheter i forhold til hvilke data SSB kan basere sin statistikk på og hvordan SSB samler inn og formidler statistikk. Omfanget av registre som kan tas i bruk som datakilder øker, samtidig som nye datakilder, som betalingssystemer og andre systemer med elektroniske spor, blir mer omfattende og dekkende. Av dette følger også kravet om en skjerpet oppmerksomhet på personvern.

SSB har som mål å holde oppgavebyrden på lavest mulige nivå, men fordi oppgavebyrdene er ujevnt fordelt oppleves disse som tunge av mange oppgavegivere. Det skal derfor arbeides for å redusere disse ved overgangen fra direkte oppgavehenting gjennom brukerutfylte skjema til utnytting av administrative registre, motivasjonstiltak, forbedring av spørreskjemaenes utforming, samordning av utvalg og tilbud til alle om elektronisk oppgaveinnhenting. Målet er at ingen som rapporterer data til det offentlige skal behøve å sende de samme dataene flere ganger for å dekke ulike statistikkbehov. Administrative registre er i dag den dominerende datakilden for SSB, og omfatter det sentrale folkeregistret i Skattedirektoratet, enhetsregistret i Brønnøysund samt Bedriftsregisteret i SSB, i tillegg til registret over Grunneiendom - Adresse - og Bygning i Statens kartverk (Statistisk sentralbyrå 2002). SSB innhenter også opplysninger fra kilder som ikke kan karakteriseres som registre, for eksempel fagsystemer i bedrifter og opplysninger i form av elektroniske spor fra betalingstjenester.

Informasjonsteknologi utgjør ryggraden i SSBs virksomhet og preger alle prosessene fra innsamling og bearbeiding av data til formidling av statistikk og analyser (Statistisk sentralbyrå 2002). Teknologi er et middel for å nå SSBs mål, samtidig som teknologi er viktig for å bedre samspillet mellom brukere og for å effektivisere interne prosesser. De siste årene har vært preget av rask utvikling av og sterk økning i bruken av Internett for spredning av data og informasjon. Det har også vært en stor økning i bruken av trådløs datakommunikasjon, noe som stiller økte krav til tilrettelegging av SSBs produkter. Resultatet av dette er økt fokus på standardformater og databaser med både data og dokumentasjon som også vil kunne gi billigere eller gratis datautveksling sammen med bruk av Internett.

Statistikk har kun verdi dersom den blir brukt (Statistisk sentralbyrå 2004), og for at den skal bli brukt er det viktig at publikum kjenner til den og vet hvor den finnes. Dette er grunnen til at også formidling av statistikk er sentralt i SSB, hvor det er viktig at man formidler nyttig og pålitelig informasjon, samtidig som denne informasjonen ikke skal inneholde så

mange detaljer at enkeltindivider kan identifiseres gjennom statistikken. For at misforståelsene av statistikken fra SSB skal bli så få som mulig ser SSB det som en viktig oppgave å informere brukere av statistikken om hvordan statistikken er laget og hvilke muligheter for feil den har. Internett er hovedkanalen for publisering av SSBs produkter, i tillegg til at SSB også produserer en rekke papirpublikasjoner.

1.2.3 IT-strategi

SSBs IT-strategi er ment å understøtte SSBs behov som statistikkorganisasjon samtidig som den skal gi rom og retning for utvikling. I denne sammenheng ser man flere nye trender og behov, som blant annet omfatter større grad av integrasjon mellom de ulike programvareløsningene. Fokus er flyttet fra klient til server, og samt at mye oppmerksomhet er flyttet fra hardwarekostnader til at man nå, og i langt større grad enn tidligere, er opptatt av programvarekostnader. De nye behovene man ser i SSB er nye brukerbehov, endringer i ressurser, endringer i kompetanse samt endringer i teknologi og verktøy. Alle disse faktorene er sammen med på å endre rammebetingelsene til SSB, og som et resultat av dette har SSB kommet frem til fem ulike retninger i IT-strategien; virksomhetsorienterte løsninger, metadatasystemer, helhetlige sikkerhetstjenester, formell og felles systemutviklingsmetode samt en bevisstgjøring rundt bruk og effekt av open source.

Med tanke på virksomhetsorienterte løsninger, herunder tjenesteorientert arkitektur, ønsker SSB å fokusere på abstraherte tjenester og logiske perspektiv definert ut fra løsningenes funksjonelle egenskaper. I fremtiden vil løsningene i større grad være meldingsorienterte, og de skal være beskrivende i forhold til funksjonaliteten de representerer. Metadata skal være maskinlesbare og alle metadata skal ha relevans. På grunn av større fokus på ressursutnyttelse er det også viktig at løsningene er presise sett fra et nytte-/kostperspektiv.

1.2.4 Rapporteringssystemer i SSB

SSBs to hovedsystemer for innsamling av data er Informasjon- og datautveksling med næringslivet (Idun) og Kommune Stat Rapportering (Kostra). Idun brukes for datautvekslingen med det norske næringslivet, mens Kostra er systemet for rapportering fra kommuner og fylkeskommuner, samt institusjoner og virksomheter under disse. I tillegg har SSB vært en aktør ved utviklingen av rapporteringskanalen AltInn, som i utgangspunktet er en rapporteringskanal for næringslivet, og har som mål å gjøre det lettere å finne, fylle ut og levere skjema til offentlige etater (AltInn n.d.). Dette kan være skjema for moms, statistikk, selvangivelser og årsregnskap. AltInn er også en rapporteringskanal for privatpersoner da enkelte personskjema benytter AltInn-motoren som rapporteringsløsning. Etatene som tilbyr skjema gjennom AltInn arbeider kontinuerlig for enklere elektroniske skjema, blant annet i samarbeid med brukere og næringslivsorganisasjoner. Etatene som er involvert i AltInn-samarbeidet er Skatteetaten, SSB,

Brønnøysundregistrene, Lånekassen, Konkurransetilsynet, Kredittilsynet, Norges Bank, Fiskeri- og kystdepartementet, Økokrim, Produksjonsregistret, Statens Innkrevingssentral, Statens Landbruksforvaltning, Husbanken og Statens Forurensningstilsyn. Alle disse etatene drifter og forvalter AltInn samlet i én sentral forvaltningsorganisasjon.

1.2.4.1 Idun

Arbeidet med Idun ble satt i gang i 2000, for å muliggjøre en nettbasert innrapportering av statistikk og nøkkeltall fra næringslivet til SSB.

Hovedmålet med prosjektet var å forenkle næringslivets arbeid med skjemautfylling (Statistisk sentralbyrå 2002). Man ønsket også å gi næringslivet bedre og mer individuelle tilbakemeldinger på rapporteringen, samtidig som man ønsket å gi tilpasset informasjon og statistikk til den enkelte bedrift. Prosjektet har hele tiden fokusert på elektronisk levering av data, og har underveis hatt et godt samarbeid med andre statsetater som Skattedirektoratet og Brønnøysundregistrene. Dette er i første rekke gjort fordi man ønsker å redusere næringslivets oppgavebyrde gjennom at en type data kun rapporteres én gang til statlige etater[2].

1.2.4.2 Kostra

Målet med Kostra har vært å bidra til å samordne og forbedre rapporteringsveien mellom kommune/fylkeskommune og staten (Statistisk sentralbyrå 2002). Kostra-prosjektet har hatt som mål å etablere felles rutiner og system for rapportering av økonomi- og tjenestedata mellom kommunen og staten. Dette, mener SSB, gir bedre muligheter og mer pålitelige data for å kunne sammenlikne kommuner og fylkeskommuner, enn hva som har vært tilfelle tidligere. Systemet inneholder data om kommunenes ressursbruk og tjenesteproduksjon, og om kommunenes innbyggere og målgrupper. Ved å sammenstille data om disse emnene skal systemet gi et felles og sammenliknbart datagrunnlag som gjør det mulig å vurdere kommunenes prioriteringer, produktivitet og dekningsgrader opp mot hverandre. Alle typer data, ressursbruk, tjenester osv. sorteres etter funksjoner som igjen skal angi de behovene kommunene skal dekke hos bestemte målgrupper. SSB tok på seg ansvaret for gjennomføring av deler av prosjektet, blant annet IT-løsningene, på oppdrag fra Kommunal- og regionaldepartementet.

Kostra har som formål å samle flere statistikkområder under én felles ramme. Dette krever felles rutiner og opplegg under hele produksjonsprosessen av statistikk, fra beslutning om datainnsamling fra en bestemt avgivergruppe, til publisering av ferdig statistikk. Per i dag rapporteres det i hovedsak på fire ulike nivå; kommunenivå, fylkesnivå, institusjonsnivå og individnivå. Hovedprosessene i Kostra omfatter et samordnet opplegg for fastlegging av innholdet i all rapportering fra kommuner og fylkeskommuner, et samlet opplegg for den delen av rapporteringen fra kommunene

[2] En mer utfyllende beskrivelse av rapporteringssystemet Idun er å finne under kapittel 3.3.1 på side 42

som går inn til SSB, publisering av nøkkeltall for kommunene, samt et opplegg for kontroll av datakvalitet[3].

Det at man har flere systemer i en organisasjon som utfører tilnærmet de samme oppgavene er uheldig på mer enn en måte. Man må arbeide med vedlikehold av flere systemer og ikke ett enkelt system som ville vært idealsituasjonen, som kan føre til økte kostnader for organisasjonen. Det at SSB opererer med to rapporteringsløsninger fører til dobbeltarbeid fordi innhenting av den samme informasjonen gjerne skjer flere ganger, og på ulike måter, samt at man gjerne fortsetter å utvikle de to systemene separat, noe som også krever økte ressurser i form av at ansatte må holdes oppdatert på flere ulike metoder, verktøy og programmeringsspråk. I tillegg får dette konsekvenser for system- og dataintegrasjonen i organisasjonen, da systemene gjerne lagrer data på ulike format og på ulike steder, det blir vanskelig å få automatisert utvekslingen av data mellom systemene, samt at ulike standarder er med på å problematisere integrasjonen, da man ikke har noen umiddelbar garanti for at disse standardene er forenelige med hverandre.

Det er viktig å se på IT som en støttefunksjon i en organisasjon som SSB, da det ikke er IT-løsninger som er organisasjonens primære virke. Spørsmål om å sentralisere IT-organisasjonen og de større IT-enhetene bør derfor ikke være en fremmed tanke, da det for en statlig organisasjon som SSB er svært viktig å være kostnadseffektiv i og med at de midlene man får budsjettert gjennom statsbudsjettet er SSBs primære inntektskilde. I denne konteksten bør man i aller høyeste grad stille seg spørsmålet om det er kostnadseffektivt med to i utgangspunktet like systemer som Kostra og Idun. Det at man har lagt ned mye arbeid og brukt store ressurser på ett system forsvarer ikke uten videre at man fortsetter å bruke ressurser på dette, dersom det skulle vise seg at det er mer lønnsomt å bruke ressursene på noe annet. Man bør likevel være åpen for å bruke eksisterende løsninger dersom disse er tilfredsstillende, og ikke forkaste godt fungerende løsninger uten grunn.

1.3 Systemintegrasjon

Uansett hvordan man ser det; det er forskjell på hvordan man skulle ønske at IT-systemene i ens organisasjon var, og hvordan de var bygget opp, kontra de faktiske forhold, det vil si hvordan disse IT-systemene faktisk er. I praksis er IT-systemer gjerne satt sammen av flere ulike teknologier, protokoller og applikasjoner, som i tillegg gjerne er koblet til hverandre på en måte som minsker fleksibiliteten ved punkt-til-punkt-forbindelser. Mellomvare skulle være selve løsningen på dette problemet, og selve ideen med mellomvare er jo nettopp å skille applikasjoner fra maskinvareplattformer, operativsystem, nettverk, protokoller og transportmekanismer som alle er i stadig endring. Vinoski (2003) hevder imidlertid at mellomva-

[3] En mer utfyllende beskrivelse av rapporteringssystemet Kostra er å finne under kapittel 3.3.2 på side 46

re, i stedet for å bli den redningen man trodde det skulle bli, har gjenskapt nettopp de samme problemene det var ment å løse. Mellomvare tilbyr riktignok abstraksjon fra maskinvareteknologiene, men ulike tilnærminger til mellomvaren krever også ulike grader av abstraksjonsnivå. Disse ulike behovene for abstraksjonsnivå gjør det samtidig vanskelig å aksessere ulike mellomvarebaserte tjenester samtidig. Vinoski mener at man best kan bruke Web services for å tilby "mellomvare-til-mellomvare"-løsninger, og som dermed vil fungere som et abstraksjonslag for dagens integrasjonsapplikasjoner.

Apache Web Services Invocation Framework (WSIF) er et av prosjektene som implementerer nettopp denne tankegangen om mellomvare-til-mellomvare gjennom å bruke Web services. WSIFs mål er i denne sammenhengen å tilby en Java API som skjuler detaljene i en Web services med flere protokoller. Tanken med dette er å la tjenester dele grensesnitt, og samtidig tillate aksess via flere ulike tilkoblingsmuligheter. WSIF er helt klart et steg i riktig retning, men som Vinoski påpeker er ikke dette nok. Den første og mest åpenbare ulempen med dette er naturlig nok at løsningen utelukkende er bygget for Java. I tillegg til at mange programmeringsspråk utelukkes av dette faktum, er det snarlig verre at dette også impliserer at man kun kan bruke protokoller tilgjengelige i Java, med mindre man er villig til å implementere støtte for andre språk selv, noe som antakelig strider med de flestes ønsker om å ta i bruk nettopp WSIF i utgangspunktet. I mange tilfeller vil det også være behov for server-tilknyttede multi-mellomvare-applikasjoner som dermed fungerer som en inngangsportal til verdifulle back-end-tjenester.

Det er ofte et ønske at ulike avdelinger skal kunne snakke sammen i sanntid oppstår behovet for informasjonsintegrasjon med ulike tilnærminger av e-business og Enterprise Application Integration (EAI) som med meldingsorientert mellomvare blir selve ryggraden i en integrasjon (Arsanjani 2002). En vanlig måte for EAI å løse et integrasjonsproblem på er å bruke adaptere for på den måten å konvertere all trafikk til kjente format og protokoller. En slik tilnærming klarer imidlertid ikke å sikre høy ytelse ved mange brukere. For å sikre høy ytelse må man bruke en multi-mellomvare-ruter som unngår konvertering så sant dette er mulig. Når en sender og en mottaker bruker samme protokoll må ruteren derfor kunne oppdage dette, og ikke konvertere meldingene mellom disse. Selv om mye arbeid gjenstår hevder Vinoski at Web services i praksis er nøkkelen til sammenkoblinger mellom IT-systemer rundt om i ulike organisasjonen, og ser på Web services som "EAI done right".

1.4 Estimering og evaluering av IT-prosjekter

Wittgenstein skrev i 1914 (Remenyi, Money, Sherwood-Smith & Irani 2000)

> Don't get involved in partial problems, but always take flight to where there is free view over the whole single great problem, even if this view is still not a clear one. — *Wittgenstein*

Dette sitatet illustrerer at det bare er i den store sammenhengen at man kan se de virkelige kostnadene og gevinstene av en IT-investering.

Mange refererer til estimering som "svart magi" (Greene 2005), og i første omgang kan man få inntrykk av at et estimat kun er basert på subjektiv gjetning. Det er ikke alltid enkelt å skulle måle eller overvåke kostnader og gevinster knyttet til IT-prosjekter, og det vil alltid være et spørsmål om hvor objektivt et estimat og en evaluering er, da den enkeltes subjektive oppfatning av prosjektet man skal estimere over og evaluere spiller en viktig rolle i prosessen.

Beslutninger og avgjørelser i virksomheter handler alltid om fremtiden, om ressursene skal brukes på det ene eller det andre prosjektet, eller på helt andre ting. Fordi beslutningene handler om fremtiden kan man bare estimere verdiene av disse. En studie gjort av The Standish Group, CHAOS, viser at et gjennomsnittlig systemutviklingsprosjekt overskrider budsjettet med 45 prosent, det går i gjennomsnitt 63 prosent over tiden og det leverer i gjennomsnitt bare 67 prosent av funksjonaliteten som er beskrevet i kravspesifikasjonen (Tockey 2004). Det er derfor viktig å bli bevisst på hvordan man estimerer kostnader, tid og kvalitet i prosjekter.

Like viktig som å gjennomføre et estimat er det viktig å forstå hvorfor man estimerer prosjekter i utgangspunktet. Grunnen til at man må estimere er som nevnt at kostnadene og gevinstene ligger inn i fremtiden, og at de derfor ikke er kjent ennå. Man kjenner ikke de nøyaktige tallene i prosjektet før produktet er ferdig utviklet og alle kostnadene og gevinstene er summert opp. Det vil derfor alltid være en viss usikkerhet knyttet til estimat over fremtidige kostnader og gevinster. Graden av usikkerhet avhenger imidlertid av hvor stor oversikt man har over prosjektet det skal estimeres over. Likevel, avgjørelsen om hvorvidt man skal starte opp et utviklingsprosjekt eller ikke må taes.

Kapittel 2

Systemutvikling, state of the art

Utviklingssyklusen for web-applikasjoner som portaler og markedsplasser blir bare kortere og kortere. Man blir på mange måter aldri helt ferdige med utviklingen av disse; man leverer versjoner som er "tilfredsstillende for øyeblikket", men som krever kontinuerlig forbedringer og videreutvikling ettersom ny teknologi og nye krav til applikasjonene kommer på banen (Agrawal, Bayardo Jr., Gruhl & Papadimitriou 2002). Nye måter å utvikle systemer på stiller nye krav til infrastrukuren man utvikler programvaren rundt, og fordi komponentene i disse systemene endres raskt og kontinuerlig bør infrastrukturen tillate løse koblinger mellom dem. Løse koblinger reduserer behovet for koordinasjon omkring utviklingsoppgaver, og man kan i større grad enn tidligere utvikle systemkomponentene raskere og i parallell (Agrawal et al. 2002).

2.1 Gjenbruk av komponenter

Mange IT-miljø ønsker å integrere sine programvaresystemer, slik at nye og gamle systemer kan samarbeide og samhandle, og man ønsker i økende grad å automatisere prosessene i organisasjonen. Mange ønsker også å fjerne seg fra "silo-tankegangen", og i større grad nærme seg komponentbaserte systemløsninger for å øke gjenbruken av komponenter i organisasjonen. Gjenbruk av programvarekomponenter er et viktig tema når man snakker om systemintegrasjon, fordi det kan være med på å effektivisere utviklings- og integrasjonsprosessen.

Meyer i Biggerstaff (1998) definerer komponenter som et programvareelement som kan brukes av utviklere som ikke kjenner utvikleren av komponenten, og hvor komponenten kan brukes i prosjekter som utvikleren av komponenten i utgangspunktet ikke kunne forutse. Gjenbruk av programvare, det vil si bruk av programvaremoduler på tvers av ulike prosjekt er et viktig ledd i en strategi for å forbedre effektiviteten i systemutviklingsprosjekter, og for å øke kvaliteten på programvaresystemene som utvikles.

Gjenbruk dreier seg i korte trekk om å finne programvaremoduler som allerede passer de kravene man måtte ha til et nytt system. Tilpasnings-

kostnadene for en modul kan for eksempel sees i forhold til i hvor stor grad den eksisterende komponenten samsvarer med komponenten man ønsker for systemet, i tillegg til hvilke krav om endringer som foreligger. Dernest må man ta i betraktning hva disse endringene vil koste.

Det er umulig å komme med bestemte påstander om hvordan ulike teknologier påvirker gjenbruk, fordi det er så mange ulike faktorer å ta hensyn til. Faktum er at det ikke finnes veldokumenterte studier som i stor målestokk måler effektene av de ulike faktorene (Biggerstaff 1998). Det finnes riktignok mindre studier, men slike er av relativt liten nytte på et generelt grunnlag.

En klar trend er at de mest vanlige teknologiene alene har liten innvirkning på graden av suksess ved gjenbruk av komponenter. I følge Biggerstaff (1998) er det en komponents domenet som er nøkkelen til gjenbruk, mens teknologi mer regnes som et verktøy i denne sammenhengen. Forholdet mellom domenet og teknologien er synergisk; begge elementer er viktige for at en løsning med gjenbruk av komponenter skal fungere. Det er også viktig å merke seg at domene-effekten er proporsjonal med størrelsen på komponenten. Små komponenter gir liten grad av gjenbruk, mens store, domene-spesifikke komponenter resulterer i en høyere grad at gjenbruk. Størrelsen på komponentene, i hvor stor grad komponentene er domene-spesifikke og prosentandelen av en applikasjon som er bygget opp av gjenbrukbare komponenter henger nøye sammen. Verden er likevel noe mer kompleks enn at man ganske enkelt kan konkludere med at man kun bør bygge store, domene-spesifikke komponenter (Biggerstaff 1998).

2.1.1 White-box- og black-box-gjenbruk

Når man vurderer gjenbruk av en komponent har utviklere stort sett tilgang til programkoden til komponenten, slik at denne kan endres for å passe 100 prosent i forhold til kravene i et nytt prosjekt (Ravichandran 2003). En slik type gjenbruk kalles gjerne white-box-gjenbruk. I de senere årene har imidlertid en annen måte å gjenbruke komponenter på blitt populær. Denne tilnærmingen kalles gjerne black-box-gjenbruk og er en klar motsetning til white-box-gjenbruk, da black-box-gjenbruk bruker programvarekomponentene som de er, uten noen form for modifikasjoner i programkoden. Selv om gjenbruk med black-box-metoden er et like gammelt konsept som white-box-gjenbruk har den på grunn av økt fokus på komponentbasert utvikling blitt mer populær i løpet av de senere årene.

Komponentbasert utvikling innebærer å utvikle programvaresystemer ved å bruke forhåndsutviklede komponenter, som i seg selv er kjørende kodebiter, og som hver for seg tilbyr det man kaller en tjeneste gjennom et veldefinert grensesnitt. Komponentbasert utvikling involverer også gjenbruk av applikasjonsrammeverk, noe som tillater at man kan samle opp komponenter og sammen la disse utgjøre et programvaresystem. Ifølge Ravichandran (2003) har pressen de siste årene kunnet rapportere om markante fordeler ved å bruke komponentbasert utvikling fremfor tradisjonelle utviklingsmetoder, inkludert white-box-gjenbruk. Analytikere har også argumentert med at komponentbasert utvikling er en viktig

bestanddel for en tjenesteorientert arkitektur med Web services. Annen litteratur på dette området indikerer tre hovedstrategier for gjenbruk; white-box-gjenbruk, black-box-gjenbruk med internt utviklede komponenter og black-box-gjenbruk med komponenter anskaffet via det åpne markedet (Ravichandran 2003).

2.1.2 Skalering av komponenter

For å kunne tilpasse en komponent til et gitt system er det viktig at komponentene er riktig skalert, dvs tilpasset slik at de tilfredsstiller de krav applikasjonen stiller til en slik komponent.

I hovedtrekk har man to typer skalering av komponenter; vertikal og horisontal skalering. Vertikal skalering går i korte trekk ut på å utvikle større og større komponenter for at gevinsten ved gjenbruk skal bli så stor som mulig, da brukeren ikke behøver skrive så mye kode selv. Likevel må man være oppmerksom på at jo større en komponent er, jo lavere sannsynlighet er det for at komponenten passer akkurat til et gitt behov, og antallet applikasjoner eller systemer hvor komponenten kan brukes synker. Selv med white-box-gjenbruk kan eventuelle mangler ved komponentene være vanskelige å endre dersom endringene gjøres av en person som ikke har gjennomgående kjennskap til komponenten. Å skalere komponenter med hensyn på egenskaper er det man kaller horisontal skalering, og er et resultat av innsnevringen av gjenbrukbare komponenter med ulike egenskaper gjennom vertikal skalering.

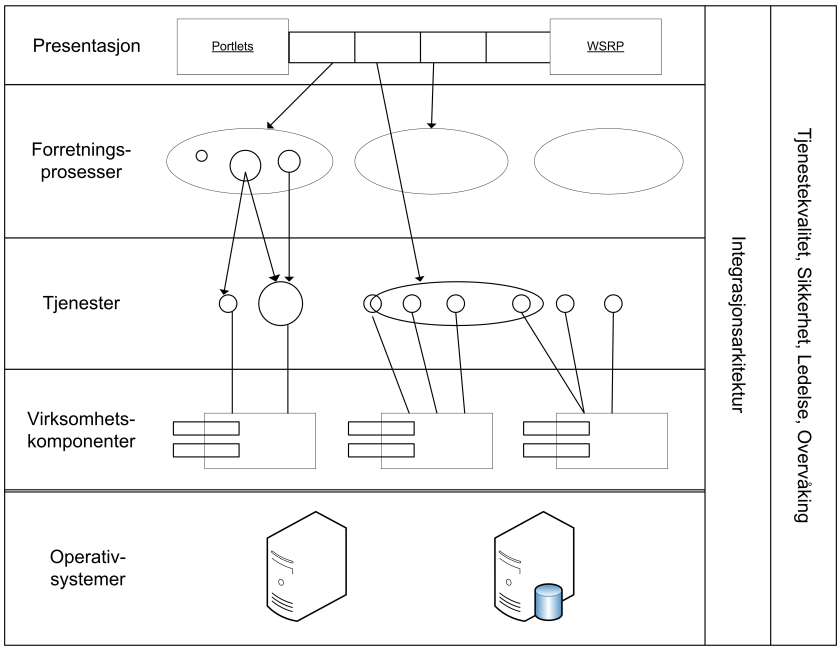
Kort sagt forsøker komponentbaserte utviklingsstrategier å optimalisere funksjonalitet i fire ulike dimensjoner; størrelse (vertikal skalering), egenskaper (horisontal skalering), ytelse og kostnader knyttet til å holde et register over komponentene. Biggerstaff (1998) argumenterer imidlertid at dette er umulig å få til, fordi det alltid vil være noe som må ofres til fordel for noe annet for å kunne lage en tilfredsstillende komponent.

2.2 Behovet for en tjenesteorientert arkitektur

Behovet for tjenesteorientert arkitektur stammer antakelig fra det økende behovet for å la en organisasjons IT-systemer bli mer smidige i forhold til blant annet forretningsmessige endringer innad i organisasjonen, og for effektivt å kunne integrere forretningsprosesser og tilhørende systemer (Huang, Li & Chao 2005). Endringene kan være forandringer i grensesnitt i meldingene komponentene utveksler, og omfatter endringer både i og utenfor organisasjonens kontekst. Den tjenesteorienterte arkitekturens endringsdyktighet anses ofte for å være en av dens største fordeler sammenliknet med andre systemutviklingsmetoder.

Tjenesteorientert arkitektur er ikke et produkt i seg selv, men snarere et sett designprinsipp som handler om å skulle tette luken mellom forretningsprosesser og informasjonsteknologi ved å ta i bruk organisasjonsrettede IT-tjenester i programvaresystemene som utvikles (Arsanjani 2004). En tjenesteorientert arkitektur kan beskrives som en måte å designe et system

på for å tilby tjenester enten til sluttbruker-applikasjoner eller andre tjenester gjennomgrensesnitt (Chung, Bylin & Davalos 2005). Tjenesteorientert arkitektur er en, ikke ny, alternativ modell til de mer tradisjonelle, tett koblede modellene vi har sett mye til de siste tiårene innen modellering og systemutvikling, og handler i stor grad om å utnytte tidligere investeringer i en organisasjon og dra nytte av disse under utviklingen av nye systemer (Newcomer 2004). En tjenesteorientert arkitektur er et rammeverk som bryter ned forretningssystemer i individuelle funksjoner og prosesser, kalt tjenester (Chung 2005), og mange mener at systemutvikling med tjenesteorienterte arkitekturer er den beste måten å møte nye utfordringer i omgivelsene på. Mange organisasjoner har derfor ikke tatt i bruk tjenesteorientert arkitektur i noen særlig grad, fordi de ikke har hatt noen klar begrunnelse for hvorfor de bør implementere nye protokoller og ny infrastruktur og styringsmiljø når eksisterende systemer fungerer tilfredsstillende. Gartner Group har anslått at mer enn 60% av alle organisasjoner vil benytte tjenesteorientert arkitektur som et ledende prinsipp innen 2008 (Arsanjani 2004).



Figur 2.1: Lagene i en tjenesteorientert arkitektur, hentet fra Arsanjani (2004)

2.2.1 Arkitekturmodell

Arkitekturmodellen i en tjenesteorientert arkitektur er en komponentbasert modell som kobler sammen funksjonalitetene til en applikasjon (se figur 2.1). Dette gjøres gjennom grensesnitt, eller veldefinerte kontrakter, mellom de ulike funksjonene arkitekturen tilbyr. Funksjonene i en tjenesteorientert arkitektur kalles tjenester og skal i størst mulig grad være uav-

hengige av hverandre. Den store fordelen med disse uavhengighetene er at en applikasjon kan overleve endringer i utviklingen av nye tjenester, samt at applikasjonen er mer smidig i forhold til forandringer i systemets omgivelser. Man omtaler dette som at tjenestene er løst koblet til hverandre. Forholdet mellom tjenester og komponenter i en slik arkitektur er at virksomhetskomponentene realiserer tjenestene, og er ansvarlige for å tilby tjenestenes funksjonalitet og vedlikeholde tjenestekvaliteten. Grensesnittene skal på sin side være definert slik at de er uavhengige av plattform, operativsystem, hardware og hvilket programmeringsspråk man har brukt under implementasjonen av tjenestene. Selv om teknisk funksjonalitet er viktig når det kommer til implementasjon av virksomhetsoperasjoner, har det liten strategisk relevans i forhold til tjenesteorientert arkitektur, og generelt sett bør ikke teknologi ha noen innvirkning på applikasjonene eller føre til avhengigheter mellom komponentene i arkitekturen (Newcomer 2004).

Løse koblinger mellom tjenester og applikasjoner gjør at det er enklere å dele data på tvers i en organisasjon, eller over et nettverk, fordi en tjenesteorientert arkitektur definerer hvordan et sett av tjenester skal plasseres i forhold til hverandre, og hvordan disse skal styres. En tjenesteorientert arkitektur kan kobles til forretningsprosessene i en organisasjon, og dermed støtte en bedre arbeidsfordeling mellom tekniske og virksomhetsorienterte medarbeidere. Tjenesteorientert arkitektur benytter også en modellbeskrivelse som kan samle nye og eksisterende systemer under ett felles system. Bruk av tjenesteorientert arkitektur kan dermed være med på å øke gjenbruk i organisasjonen, redusere kostnader og forbedre muligheten til å endre og videreutvikle nye og gamle IT-systemer.

Tjenesteorientert arkitektur er i tillegg til tjenestene i seg selv basert på følgende fire begrep; applikasjon, tjeneste, tjenesteregister og tjenestebus. Selv om det er applikasjonen som er eier av forretningsprosessene, tilbyr tjenestene funksjonalitet som applikasjonen og andre tjenester kan ta i bruk. Tjenester defineres gjerne som en implementasjon som tilbyr forretningslogikk, data, en kontrakt som spesifiserer tjenestens funksjonalitet og bruk, og ett grensesnitt som eksponerer denne funksjonaliteten til potensielle brukere. Tjenesteregisteret lagrer alle kontraktene inneholdt i en arkitektur, mens tjenestebus'en kobler applikasjonene og tjenestene sammen (Krafzig, Banke & Slama 2005). Tjenesteorientert arkitektur kan derfor defineres slik:

> A Service-Oriented Architecture (SOA) is a software architecture that is based on the key concepts of an application frontend, service, service repository and service bus. A service consists of a contract, one or more interfaces, and an implementation.
>
> — *Krafzig et al. (2005)*

Applikasjonen er det aktive elementet i tjenesteorientert arkitektur, siden det er denne som leverer verdien av arkitekturen til sluttbrukerne. Det er likevel viktig å huske at det er tjenestene som gir arkitekturen struktur, og selv om tjenestene forblir uforandret, er det stor sannsynlighet

for at applikasjonene endres i løpet av tjenestens levetid, som følge av endringer i organisasjonens forretningsprosesser. På bakgrunn av dette er også livssyklusen til applikasjonene langt kortere enn tjenestenes livssyklus, og tjenestene regnes derfor for å være hovedbestanddelene i en tjenesteorientert arkitektur.

2.2.2 Karakteristika ved tjenester

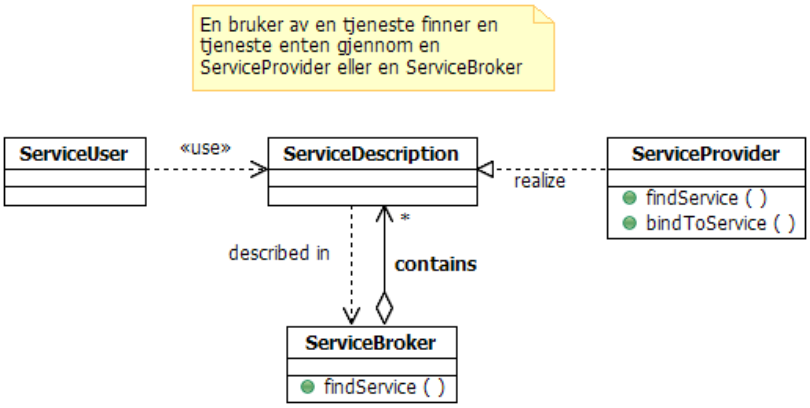
Fra et virksomhetsperspektiv er tjenester aktiva som korresponderer med organisasjonens virkelige aktiviteter eller gjenkjennbare funksjoner som kan aksesseres i henhold til reglene som er gjeldende for den aktuelle tjenesten. Disse reglene angir blant annet hvem eller hva som er autorisert for tilgang til tjenesten, i hvilke(t) tidsrom tjenesten er tilgjengelig, kostnadene ved å bruke tjenesten, pålitelighet for tjenesten, sikkerhetsnivået og ytelsen. Fra et teknisk perspektiv er tjenester grovkornede, gjenbrukbare IT-aktiva med veldefinerte grensesnitt som tydelig skiller tjenestens eksterne grensesnitt fra tjenestens tekniske implementasjon. Dette skillet gjør at bruker og leverandør av tjenesten separeres, slik at disse kan utvikle seg individuelt, så lenge tjenesten forblir uforandret.

Tjenester er satt sammen av moduler, og det modulære designet er viktig for tjenesteorienterte arkitekturer. I denne konteksten ser man på en modul som et sett av programvarekomponenter som til sammen utgjør en verdi for en bruker av en tjeneste. Modulen skal utføre kun én funksjon, men denne skal til gjengjeld utføres fullstendig. Moduler bør derfor være enkle komponenter.

Tjenester kan beskrives med primære og sekundære karakteristika (Newcomer 2004). De primære karakteristika av tjenester med tanke på design, implementasjon og styring er løse koblinger, veldefinerte tjenestekontrakter, meningsfullt innhold for brukere og at de er standardbaserte. En tjeneste bør i tillegg oppfylle så mange av de sekundære karakteristika som mulig for å kunne tilby organisatoriske og tekniske fordeler. De sekundære karakteristikkene er at tjenestene skal forutsette avtaler mellom tjenestenivåene, tjenestene skal være dynamiske, de skal være mulige å oppdage for potensielle brukere samt metadata-baserte, implementasjonen av en tjeneste skal være uavhengig i forhold til andre tjenester, tjenestene bør være tilstandsløse, og sist men ikke minst bør tjenester designes med tanke på ytelsen de skal tilby.

Viktigheten av kontrakter i en tjenesteorientert arkitektur kan neppe uttrykkes tydelig nok. En formell kontrakt er det elementet som tillater en formalisering av systemet og systemgrensene, som å minimalisere avhengighetene mellom tjenestene, øke anvendeligheten og gi brukere et godt grunnlag for å kunne velge mellom ulike tjenester. Kontraktene er gjerne mer verdt enn implementasjonene, fordi kontraktene også inneholder informasjon om organisasjonen som er leverandør av tjenesten. Tjenester bør også, i størst mulig grad, være basert på åpne standarder, og den bør være forutsigbar noe som blant annet har å gjøre med tjenestens responstid, gjennomstrømning, tilgjengelighet og feiltetthet. Tjenester skal publiseres på en slik måte at de kan oppdages og brukes uten hjelp

fra leverandøren. Kontraktene skal også bruke metadata for å definere mulighetene og begrensningene med tjenesten. Figur 2.2 viser den logiske modellen av en tjenesteorientert arkitektur.



Figur 2.2: Logisk modell av en tjenesteorientert arkitektur, hentet fra Arsanjani (2004)

Tjenester bør implementeres slik at avhengighetene mellom dem blir så små som mulig. Viktigst er det likevel at tjenester er selvstendige, slik at de kan samhandle med andre tjenester uten nødvendige indre avhengigheter og uten å ha noen felles tilstand. Det å skulle designe og implementere tjenester som støtter flere måter å bli kallet på, inkluderer støtte for asynkrone meldinger, batch prosessering og ulike former for spørringer og tilbakemeldinger. Baglietto, Maresca, Parodi & Zingirian (2004) hevder i tillegg at alle operatører skal kunne ha tilgang til tjenesten uavhengig av operatørens og tjenestens infrastruktur, tjenestene skal tillate kommunikasjon både mellom menneskelige operatører og på tvers av virksomhetssystemer. Operatører skal derfor også kunne samhandle, selv om deres teknologi, interne systemer og organisasjoner er forskjellige. Det skal være mulig å oppdatere dataformat og datautvekslingsformer raskt og billig, det skal være mulig å utveksle meldinger og transaksjoner med sensitive personopplysninger og sist men ikke minst skal ikke brukerne av en tjeneste tvinges til å bruke denne for å koble seg opp mot andre nettverkstjenester eller applikasjoner.

Tjenester er definert ut fra meldingene de utveksler, enten med andre tjenester eller med sluttbrukere, og de skiller seg derfor klart fra blant annet objekter og prosedyrer kjent fra objektorientert systemutvikling (Newcomer 2004). Selv om designet til tjenesteorienterte arkitekturer, som navnet tilsier, er tjenesteorientert, utelukker ikke dette at de individuelle tjenestene innad i arkitekturen kan være utviklet med et objektorientert design. En tjenesteorientert arkitektur kan godt være objekt-*basert*, men ikke objektorientert som helhet. Vanlige og eksisterende modelleringsdisipliner kommer til kort når de er utviklet uavhengig av hverandre. I tjenesteorientert arkitektur løses dette med tre hovednivåer av abstraksjon; operasjoner, tjenester og forretningsprosesser. Operasjonene er de logiske delene av en

prosess med et spesifisert og strukturert grensesnitt, og hver operasjon leverer en strukturert melding. Tjenestene representerer de logiske grupperingene av operasjonene, mens forretningsprosessene er et kjørende sett av handlinger eller aktiviteter som utføres i forhold til bestemte mål knyttet til organisasjonen.

Tjenester kan representere mange problemområder som kan grupperes i virksomhetsdomener og tekniske domener. De vanligste typene av tjenester er de som tilbyr teknisk funksjonalitet. Virksomhetstjenester er vanskeligere å implementere enn tekniske tjenester, men er av langt større verdi for organisasjonen, både internt og eksternt. Å utvikle tjenester som representerer en bestemt forretningsprosess har langsiktig verdi ved bruk av tjenesteorientert arkitektur fordi tjenesten vil ta for seg arbeidsflyten i organisasjonen.

2.2.3 Infrastruktur

Tjenesteorientert arkitektur er ikke en teknologi i seg selv, men bør sees som en metode for å designe og dokumentere arkitekturen av tjenester, samt en metode for å dokumentere forhold og avhengigheter tjenestene imellom. Tjenestene skal være levert på et passende abstraksjonsnivå, med en passende struktur og med en generalisering som er fornuftig både for bruker og leverandør. Tjenesteorientert arkitektur som disiplin er med på å redusere belastningen på applikasjonene ved å bruke et sett av tjenester for å nå et gitt mål og minimalisere endringene ved at brukere kan bytte leverandører og leverandører kan bytte tjeneste-implementasjon.

Tjenesteorienterte arkitekturer er uavhengige av hvilken type nettverksforbindelse som benyttes, og tjenester er uavhengige av hvilken type transportmekanisme som benyttes for brukere å koble opp mot tjenestene. I praksis betyr dette at det opprettes innstillinger for å kunne nå tjenesten via flere ulike typer transportmekanismer, noe som vanligvis opprettes ved behov. Tjenesteorientert arkitektur gjør det også mulig å tilby nye tilkoblingsprotokoller ettersom slike utvikles. Ideelt sett bør innstillingene ikke være språkspesifikke, i det dette bryter regelen om introspeksjon.

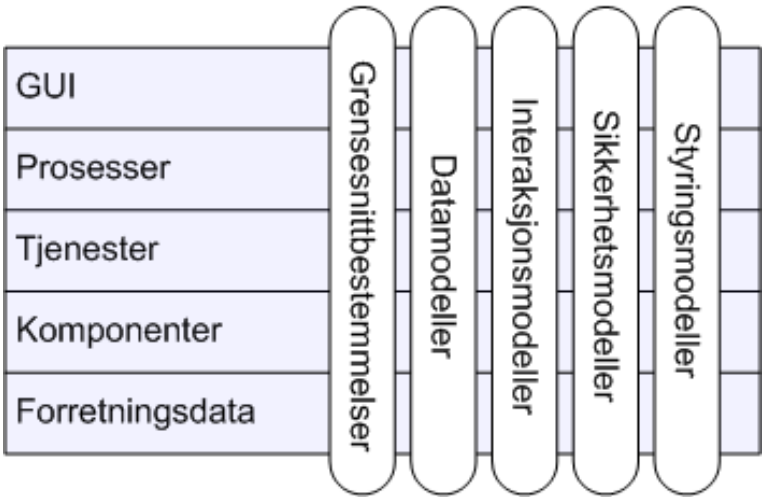
I og med at klienten som bruker tjenesten ikke er avhengig av å vite hvor en tjeneste fysisk sett er lokalisert, gir dette stor grad av fleksibilitet i implementasjonen av tjenester, da instanser av tjenester kan spres over flere maskiner i et cluster. Dette gjør at arkitekturen er mer fleksibel, enklere å holde ved like, billigere i drift og mer resistent i forhold til uforutsette hendelser. Ideelt sett opererer tjenester uavhengig av både systemet og plattformen de kjøres på, noe som først og fremst er en fordel for organisasjoner som bruker flere plattformer i sin infrastruktur. For å få en tjenesteorientert arkitektur til å fungere tilstrekkelig godt er det likevel viktig å sikre at man har nok båndbredde i nettverket slik at man kan tilby god dataflyt. Det er også viktig at man til enhver tid har ledige nettverkstilkoblinger for å øke tilgjengeligheten til de ulike tjenestene man tilbyr, og å ha muligheten til å dedikere maskiner som utelukkende tar seg av tjenestene og funksjonaliteten rundt disse.

Før implementasjon av en tjenesteorientert arkitektur bør man vurdere

om organisasjonen har den infrastrukturen som kreves for å tilby økt ytelse, et krav som kommer som en følge av at tilkoblingene til organisasjonen distribueres. Sikkerhetsaspektene ved tjenesteorientert arkitektur er også en svært sentral problemstilling ettersom man utsetter organisasjonen for økte risiki ved å åpne tilkoblinger til nettverket. Til slutt, og før man begynner implementasjonen av en tjenesteorientert arkitektur, er det viktig å ha en konkret plan for hvordan organisasjonen skal håndtere og styre tjenestene sine.

2.2.4 Systemarkitektur

Det er viktig å være oppmerksom på at en tjenesteorientert arkitektur ikke er en ny form for applikasjonsarkitektur (Newcomer 2004). Tjenesteorientert arkitektur er rettet mot å definere og levere forretningstjenester og tekniske tjenester som tilbyr organisasjonen funksjonalitet og som er gjenbrukbare på tvers av flere brukergrupper, applikasjoner og kanaler. Tjenesteorientert arkitektur fokuserer derimot ikke på hvordan disse tjenestene kombineres i applikasjonene og presenteres for de ulike brukergruppene. Tjenesteorientert arkitektur fokuserer i stor grad på abstraksjon på tjenestenivå, fordi de kun behandler tjenester og de definerer alle viktige elementer i en tjeneste, se figur 2.3.



Figur 2.3: Tjenesteabstraksjoner, hentet fra Newcomer (2004)

Lagene i abstraksjonen representerer kjente lag i virksomhetsarkitekturen; brukergrensesnitt (GUI) for brukere av virksomheten, forretningsprosesser som koordinerer aktivitetene i virksomheten, tjenester som modellerer og definerer individuelle aktiviteter som gjenbrukbare og teknologinøytrale ved å bruke Web services, se Kapittel 2.4, komponenter og objekter som brukes for å implementere og utføre tjenester, virksomhetsdata lagret i databaser eller liknende. Kolonnene i abstraksjonen representerer interesseområder som appellerer til ulike lag i arkitekturen. De vanligste abstraksjonene på tjenestenivå er grensesnittdefinisjoner, datamodel-

len, interaksjonsmodellen, sikkerhetsmodellen og styringsmodellen. Alle tjenester innenfor et tjenestedomene bør kommunisere ved hjelp av et felles vokabular, slik at disse tjenestene enkelt kan utvikles og knyttes opp mot hverandre. Gjenbrukbare tjenester er tjenester som kan benyttes på flere nivåer i organisasjonen. Eksempler på slike tjenester vil typisk være tjenester som inkluderer datatransformasjon, dataaksess, revisjon, logging, identitetskontroll osv. Disse gjenbrukbare tjenestene er spesielt viktige fordi de er med på å redusere risiki knyttet til utvikling av ny programvare og nye programvaresystemer.

Kompleksitet er et faktum i IT-verdenen, og det er knyttet store utfordringer til kompleksitet ved utvikling av nye applikasjoner, under arbeid med å erstatte eksisterende applikasjoner og for å holde tritt med krav om vedlikehold og forbedringer. En innføring av tjenesteorientert arkitektur lover enklere, mindre kompleks og i større grad gjenbrukbar funksjonalitet. Tjenester legger også grunnlaget for flere strategiske løsninger; rask applikasjonsintegrasjon, automatiserte forretningsprosesser, samt flere tilgangskanaler til applikasjoner, inkludert både stasjonære og mobile enheter.

2.3 Integrasjon med tjenesteorienterte arkitekturer, suksesshistorier

2.3.1 Guardian Life Insurance

I 2000 måtte det amerikanske forsikringsselskapet Guardian Life Insurance tenke gjennom strukturene rundt sine applikasjoner, som var utviklet med liten oppmerksomhet i forhold til organisasjonens mål. Organisasjonen hadde ikke opprettet noen standarder for å utvikle eller koble sammen applikasjoner, ei heller noen standard for gjenbruk av kode. Forsikringsselskapet bestemte seg for å utvikle felles tjenester der dette var mulig, for med det å redusere kompleksiteten rundt applikasjonene, fordi de var av den oppfatning at en tjenesteorientert arkitektur var den eneste måten de kunne få de ulike applikasjoner til å samhandle på (Gruman 2005). Med en tjenesteorientert arkitektur kunne de dessuten fokusere på å utvikle nye applikasjoner i stedet for å arbeide på de gamle, selv om gjenbruk var et viktig element i arbeidet.

Den tjenesteorienterte arkitekturen til Guardian Life Insurance er bygget opp rundt en håndteringsmekanisme for de virksomhetskritiske tjenestene, som er en samling av J2EE verktøy som fungerer som mellomvare, og en IBM CICS/MQSeries Message Bus for å håndtere forespørsler (Gruman 2005). Forespørslene kommer fra ett av tre client systemer, som i virkeligheten er en web portal, et CRM system[1] og et interaktivt telefonsystem brukt av kunder, samt forespørsler fra applikasjonene selv. Bakgrunnen for dette valget var at selskapet mente at en sentral mekanisme for håndtering av de virksomhetskritiske tjenestene var den beste måt-

[1]CRM (Customer Relationship Management) system er verktøy for å håndtere en virksomhets kunder og relasjonene til disse

en de kunne øke gjenbruket av applikasjoner og kode i organisasjonen på (Gruman 2005).

Fordi forsikringsselskapet i stor utstrekning bruker applikasjoner som kjører sentralt på stormaskiner møtte IT-teamet i organisasjonen en utfordring i forhold til å eksponere disse applikasjonene slik at de kunne bli brukt som tjenester. I stedet for å omskrive disse applikasjonene valgte selskapet å bruke Enterprise Java Beans (EJB) for å utføre oversettelser utenfor applikasjonen mot andre tjenester. Guardian Life Insurances erfaring var i hovedsak at en omlegging til en tjenesteorientert arkitektur først og fremst er en kulturell forandring. Det er viktig å være klar over at det er en stor utfordring å lykkes og at en slik integrasjon krever en god del koordinasjon og planlegging. Estimatene til forsikringsselskapet tilsier at de har spart rundt 30 prosent av budsjettet for applikasjonsutvikling.

2.3.2 Transamerica

Ett av løftene til den tjenesteorienterte arkitekturen er å gjøre det mulig for selskaper å påvirke eksisterende systemer som et sett av grunnleggende, gjenbrukbare byggeklosser som kan settes sammen, for dermed å kunne lage nye prosesser og applikasjoner raskt og rimelig (Erlanger 2005). Det amerikanske forsikringsselskapet Transamerica utveksler store mengder data med ulike distributører. Selskapet oppdaget at for å fortsatt være konkurransedyktige måtte de kunne tilby sine partnere sanntidsaksess til sine systemer, noe som i seg selv er en kompleks operasjon. For å kunne gjøre dette hadde Transamerica behov for ulike valideringsmekanismer for representanter av ulike selskaper tilknyttet Transamerica (Erlanger 2005). Med denne kompleksiteten innså selskapet at Web services og en tjenesteorientert arkitektur var et naturlig valg, da de hadde behov for en løsning som var både tett integrert og løst koblet.

Transamerica hadde store mengder forretningslogikk i sine systemer, og i stedet for å omskrive denne logikken ønsket de å opprette noen bakenforliggende tjenester som kunne eksponere denne forretningslogikken slik at den var tilgjengelige for flere ulike applikasjoner, prosesser og kanaler, enten det dreide seg om batch-prosesser, sanntidsprosesser over nettet, interne applikasjoner eller gjennom andre systemer. For å støtte alle disse aksessmulighetene måtte hver tjeneste ha evnen til å kunne motta forespørsler gjennom MQSeries og Java Messaging Service (JMS) i tillegg til vanlige SOAP[2]-meldinger. Disse tjenestene kunne dermed settes sammen til større grupper tjenester, skreddersydd for en type behov i forhold til kommunikasjonskanaler og forretningspartnere. Sluttbrukerapplikasjonene bruker et XML Schema, som kan transformeres til et passende format i forhold til hva kundenes systemer krever.

Ledelsen i Transamerica råder, i likhet med Guardian Life Insurance, andre som skal benytte tjenesteorienterte arkitekturer i sine organisasjoner til å gjøre så mye planlegging og forberedelser som mulig i forkant av implementasjonen, og sier at dersom de skulle gjort det en gang til ville

[2]Simple Object Access Protocol

de brukt mer tid på prototyping og testing, samt å sette opp arkitekturen og standardene rundt denne.

2.3.3 IT-arkitektur i offentlig sektor

En arbeidsgruppe i Danmark har laget en rapport om IT-arkitektur i offentlig sektor i regi av Ministeriet for Videnskab Teknologi og Udvikling og Det Koordinerende Informationsudvalg kalt "Hvidbog om IT-arkitektur"(Gartner Consulting 2004). Arbeidsgruppen skriver blant annet at digital forvaltning i stor grad handler om å få de offentlige IT-systemene til å samarbeide, slik at de ulike myndighetene i landet skal ha anledning til å bruke hverandres data på en slik måte at borgere, organisasjoner og saksbehandler ikke skal måtte avgi og kontrollere samme informasjon flere ganger. Arbeidsgruppen skriver videre at dette krever felles definisjoner av data og lik håndtering av sikkerhet og brukere, for å nevne noe.

Danmarks regjerings moderniseringsprogram har satt seg som mål å forbedre sin service overfor borgere og næringsliv, samtidig som de skal øke effektiviteten i offentlig administrasjon. Danmarks Statistikk utførte i 2002 en undersøkelse av offentlig IT-bruk som viste at noen av de viktigste utfordringene digital forvaltning står overfor er innen nettopp IT-arkitektur. Så mange som 7 av 10 myndigheter savner felles offentlige løsninger og infrastrukturer, og omtrent like mange savner en felles standard for datautveksling (IT og Telestyrelsen 2002).

Hovedanbefalingen fra arbeidsgruppen er at offentlig sektor må ta et mer aktivt ansvar for sin egen IT-arkitektur, samtidig som gruppen mener at det bør etableres en felles IT-arkitekturramme for planlegging av offentlige IT-systemer for å i første rekke sikre interoperabilitet. Arbeidsgruppen peker på noen elementer som de mener denne IT-arkitekturrammen bør inkludere. Noen av disse er et felles valg omkring standarder og infrastruktur, en felles metoderamme med prosesser, begreper og beskrivelsesstandarder for IT-arkitekturen og en felles koordinering av arbeidet.

Gartner Consulting mener at man så tidlig som mulig bør begynne arbeidet med å heve den alminnelige kompetansen innen tjenesteorienterte arkitekturer og benytte tjenesteorienterte prinsipp i utviklingsprosjekter. De anbefaler i tillegg at tilkobling mot felles tjenester bør være en del av de felles kravene man stiller til offentlige IT-systemer. Gartner Consulting påpeker også at den tjenesteorienterte arkitekturen ikke bør benyttes for applikasjoner med begrenset utbredelse og levetid, fordi forretningslogikken i slike applikasjoner verken blir gjenbrukt eller endret i løpet av levetiden til applikasjonen. En tjenesteorientert arkitektur bør heller ikke benyttes for applikasjoner med enveis asynkrone interaksjoner, hvor de løse koblingene i en tjenesteorientert arkitektur både er unødvendige og uønskede (Gartner Consulting 2004).

Gartner peker samtidig på det faktum at en tjenesteorientert arkitektur ikke på noen måte vil løse alle problemene i offentlig sektor knyttet til IT-arkitektur, og formulerer to viktige spørsmål som bør stilles når man vurderer å innføre en tjenesteorientert arkitektur i de offentliges IT-systemer;

- *Hvordan omsettes ønsket om en tjenesteorientert arkitektur til konkrete retningslinjer som kan anvendes overfor systemleverandører?*
- *Hvordan skal strategien utformes i forhold til eksisterende systemer?*

Både Gartner og arbeidsgruppen fra "Hvidbog for IT-arkitektur"anbefaler imidlertid at tjenesteorienterte arkitekturer tas i bruk i offentlige IT-systemer, da en slik arkitektur kan løse noen av utfordringene offentlig sektor står overfor, spesielt med tanke på økt kompleksitet i IT-arkitekturen og derav komplekse utviklingsprosjekter (Gartner Consulting 2004).

2.4 Tjenesteorientert arkitektur med Web services

En beskrivelse av ulike teknologier for å lealisere en tjenesteorientert arkitektur er beskrevet i Tillegg A fra side 111. I dette kapittelet vil jeg kun beskrive tjenesteorientert arkitaktur realisert med Web services.

En av de største fordelene med en tjenesteorientert arkitektur realisert gjennom Web services er at den tilbyr bedre ansvarsfordeling mellom organisasjonsanalytikere og tjenesteutviklere. Arbeidet med å integrere eksisterende og nye applikasjoner ved å bruke tjenesteorientert arkitektur involverer også å skulle definere et grunnleggende interoperabilitetslag samt et lag som gjenspeiler tjenestekvaliteten i organisasjonen for å sikre egenskaper og funksjoner i de gjeldende applikasjonene, som sikkerhet, pålitelighet og transaksjonshåndtering. En tjenesteorientert arkitektur med Web services tilbyr utvikling av tjenester som involverer virksomhetsfunksjonene som er aksesserbare fra andre tjenester. For å kunne utvikle en god integrasjonsarkitektur med Web services er det først og fremst viktig å forstå nivåene i en integrasjon med Web services, og designe med tanke på virksomhetskomponenter og tjenester i en kontekst av komponentbasert utvikling og integrasjon (Arsanjani 2002, Arsanjani, Hailpern, Martin & Tarr 2003).

2.4.1 Web services plattformen

En tjenesteorientert arkitektur basert på Web services har fordeler i at den er basert på åpne standarder, tilbyr interoperabilitet for ulike løsninger, den støtter interorganisatorisk integrasjon, Web services plattformen tilbyr fasiliteter for at tjenestebrukere og tjenesteleverandører kan integrere uavhengig av underliggende programvare, som igjen fremprovoserer forretningsregler og poliser som valideringsregler, sikkerhetsnivå, tjenestestyring og tjenestekontrakter, og lar arkitekturen håndtere virksomhetenes felles forretningskrav, se figur 2.6 på side 29.

I sin videste forstand er Web services plattformen basert på åpne standarder som er produktnøytrale, teknologinøytrale og mellomvarenøytrale (Newcomer 2004). Elementer i plattformen er tjenestekontrakter, lagringsplass for tjenestekontrakter (såkalte tjenesteregistre), en tjeneste for å finne og registrere andre tjenester, sikkerhetsmekanismer, datahåndtering, kommunikasjonshåndtering, støtte for flere protokoller og transportfasiliteter,

kvalitetsattributter for tjenesten, styring og kontroll over tjenesten, og støtte for flere programmeringsspråk og programmeringsgrensesnitt.

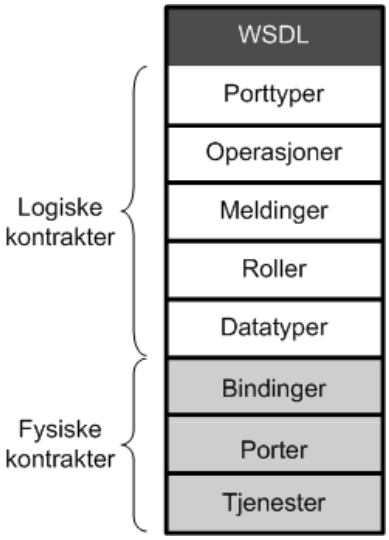
Selv om Web services plattformen er basert på standarder, kan det ta en stund før et komplett sett av standarder er ferdig utviklet og utprøvd (Newcomer 2004). Fordelen er likevel at en tjenesteorientert arkitektur med Web services er komponentbasert, slik at man kan utvikle komponenter litt etter litt. Web services plattformen bør unngå å inkludere alle gjenbrukbare tekniske tjenester, fordi enhver ny gjenbrukbar tjeneste i en Web services plattform gjør tjenesten mer komplisert, vanskeligere å vedlikeholde og vanskeligere å utvide ettersom kravene til tjenesten endres. I stedet bør gjenbrukbare tekniske tjenester bygges på toppen av plattformen. På denne måten kan man endre gjenbrukbare tekniske tjenester uten å endre plattformen. Et annet prinsipp med plattformen er at den innlemmer vanlige egenskaper som å automatisk støtte opp om forretningsregler eller flytte kompleksitet ut av applikasjonslaget og inn i plattformen. Dette prinsippet er også med på å forklare hvorfor enkel pålogging, rollebasert aksesskontroll, overvåking med loggføring, pålitelig meldingsutveksling og transaksjonshåndtering er inkludert i Web services plattformen.

2.4.2 Kontrakter

Hver tjeneste i en tjenesteorientert arkitektur med Web services har en veldefinert kontrakt som definerer hva tjenesten gjør, og skiller klart mellom de aksesserbare grensesnittet og den tekniske implementasjonen. Noen elementer i kontrakten appellerer til hele tjenesten, andre appellerer kun til operasjonene som utgjør tjenesten. Elementene bør være maskinlesbare slik at verktøy kan utnyttes for å utvikle, kjøre og styre aktivitetene, og elementene i en kontrakt vil typisk være et tjenestenavn, et versjonsnummer, pre-betingelser og en klassifisering av tjenesten. Klassifiseringen vil i de fleste tilfeller være nøkkelord for å beskrive de forretningsdomenene som tjenesten støtter. Hver operasjon har et operasjonsnavn, pre- og postbetingelser, en profil for input-data, en profil for output-data, en interaksjonsprofil, feilhåndtering og unntakstilfeller, en sikkerhetsprofil, transaksjonsprofil og gjenopprettingsrutiner, samt kontrakter om tjenestestyring (Newcomer 2004).

En kontrakt kan enten defineres ved hjelp av Web Services Definition Language (WSDL), XML Schema og WS-Policy framework, eller det kan defineres implisitt på bakgrunn av godtatt in- og output, samt forretningsaktivitetene som tjenesten implementerer. Man bør uansett strebe etter å gjøre kontraktene så abstrakte og generelle som mulig, slik at alternative implementasjoner kan tilbys når det måtte være nødvendig. Det er nettopp gjennom tjenestekontraktene man oppnår interoperabilitet og integrasjon, og begge partene i en integrasjon må forstå den samme definisjonen av tjenesten, inkludert navnet, meldingene, utvekslingsformatene, datatypene og annen assosiert informasjon. WSDL er det ideelle valget som definisjonsspråk fordi det er standardbasert, utvidbart, det bygger på XML Schema og skiller klart mellom logiske og fysiske kontrakter.

Kontrakten er distribuert av bruker og leverandør av tjenester og



Figur 2.4: Oppbygningen av en tjenestekontrakt, hentet fra Newcomer (2004)

har funksjonalitet for at bruker og leverandør skal kunne utveksle data i et format begge er fortrolige med. Se figur 2.4 for oppbygging av en tjenestekontrakt, hvor den logiske delen beskriver grensesnittet som er uavhengig av transport, dataformat og programmeringsspråk, mens den fysiske delen av kontrakten definerer bindinger til transporttyper og dataformat. Det er verdt å merke seg at flere fysiske kontrakter kan defineres for hver logiske kontrakt (Newcomer 2004).

En tjenesteorientert arkitektur må kunne definere, parse, validere og transformere meldinger. Til sammen utgjør dette arkitekturens første datamodell og databehandlingsmuligheter. XML Schema er en god teknologien for å representere tjenestenes datamodell i tjenesteorientert arkitektur, fordi den er åpen, standardisert og utvidbar.

2.4.2.1 Web Service Definition Language, WSDL

Kontrakten som beskriver hva en tjeneste krever av og tilbyr til brukeren av tjenesten bør beskrives i form av et WSDL-dokument. Et slikt dokument består av funksjonaliteten som tjenesten tilbyr, en beskrivelse av meldingen som aksepteres inn til tjenesten og hvilket format denne meldingen skal være på, samt protokollen som brukeren må benytte for å kunne ta i bruk tjenesten. Spesifikasjonen av WSDL-vokabularet definerer et XML-vokabular for igjen å definere kontrakten mellom en bruker og en leverandør i termer av meldinger. Under meldingsutveksling sender en bruker en melding til en leverandør og etter at leverandøren har prosessert meldingen sender leverandøren en responsmelding tilbake til brukeren. Det er god praksis å designe grovkornede grensesnitt som minimerer antallet utvekslinger for en transaksjon. Det er sammen med WSDL-kontrakten at Web services tilbyr en mekanisme for å dynamisk bygge en

proxy som tillater tilkobling til tjenesten ved bruk av WSDL.

2.4.2.2 Simple Object Access Protocol, SOAP

XML er basisen for all Web services teknologi. Web services bruker SOAP-protokollen for å overføre data, og denne definerer en standard for hvordan meldinger skal representeres og kalles (Ma 2005). SOAP definerer også datatypene som skal brukes dersom man benytter "remote procedure calls" (RPC). SOAP uttrykkes som XML, og en SOAP-melding består av tre hoveddeler; en konvolutt som identifiserer XML som en SOAP-melding, et hode som holder informasjon og transaksjonsstyring, routing, sikkerhet, prosessering med mer, og en kropp som inneholder de faktiske dataene i meldingen. SOAP-protokollen har ikke klart å definere en fullstendig og entydig datakommunikasjon, og interoperabilitet mellom tjenester er fortsatt et mål det viser seg at det kan være vanskelig å nå. Tjenesteorientert arkitektur krever imidlertid ikke bruk av SOAP. Det finnes også Java API som gjør det enkelt å overvåke en Web service via et Java-program.

2.4.3 Registre

Universal Description Discovery and Integration (UDDI) definerer en SOAP-basert programmeringsprotokoll for å registrere og avdekke Web services. Det finnes to typer UDDI-registre; private og offentlige. De offentlige registrene er en distribuert tjeneste som utveksler data med hverandre, private registre er derimot kun tilgjengelige for en definert gruppe brukere, eller internt i en organisasjon. UDDI er også et tiltak for å etablere en standard business-to-business integrasjon og definerer et sett med standard grensesnitt for å få aksess i databaser av Web services (Newcomer 2004). UDDI-spesifikasjonene definerer en metode for å publisere og avdekke informasjon om Web service, og oppretter et plattformnøytralt rammeverk for å beskrive tjenester, avdekke forretninger og å integrere forretninger via Internett.

2.4.4 Interaksjonsmønstre

Virksomheter utveksler informasjon på mange ulike måter, og for å imøtekomme dette må arkitekturen støtte mange ulike interaksjonsmønstre. De mest vanlige er query/reply (spørring/svar), query/response (spørring/tilbakemelding), enveis kommunikasjon og publish/subscribe (tilbud/abonnement). SOAP og HTTP er synkron og basert på spørring/tilbakemelding, fordi det kreves at både bruker og leverandør er online på samme tid, og at data kan utveksles i begge retninger. Asynkron kommunikasjon kan oppnås på to måter; ved å bruke proxyer som abstraherer kommunikasjonslaget og metadata som hentes automatisk av tjenesteplattformen, eller ved at rutiner eksplisitt utfører all ordning av data som kreves for å legge til og fjerne meldinger fra køen. I de aller fleste tilfeller er det mest tilrådelig å bruke proxyer, nettopp fordi disse kan abstrahere kommunikasjonslaget og utføre ordning av data, og konvertere objekter til

meldinger og parse meldingsbuffere (Newcomer 2004). Standardene WS-Reliability og WS-ReliableMessaging garanterer levering av meldinger.

Det er en stor fordel at Web services benytter åpne standarder. Det er disse åpne standardene som utgjør basisen i en Web service og gjør det mulig for en gitt Web service å kommunisere med en hvilken som helst annen Web service. De åpne standardene som brukes av Web services er transportprotokoller som HTTP, FTP og SMTP, meldingsprotokollen SOAP, WSDL for å beskrive grensesnitt (kontrakter), samt registerprotokoller som UDDI og lagringsprotokoller som ebXML.

For tiden kommuniserer alle Web services over HTTP. FTP og SMTP betegnes som alternative kommunikasjonskanaler, men brukes ikke i noen stor utstrekning nettopp fordi de krever kommunikasjon via en applikasjon, noe som strider mot prinsippet om at Web services, og tjenester i en tjenesteorientert arkitektur generelt, skal være uavhengige av typen transport som velges (Newcomer 2004). Uansett må man nok regne med at HTTP vil fortsette å være den mest brukte kommunikasjonskanalen også i tiden som kommer, fordi HTTP-meldinger enkelt kan passere brannmurer. Dette er spesielt gunstig for virksomheter som benytter seg av tjenester som fysisk er plassert i andre virksomheter, eller på et annet sted i samme virksomhet.

2.4.5 Interoperabilitet

En tjeneste er en lokasjon i et nettverk med en maskinlesbar beskrivelse av meldingene den kan motta og returnere. En tjeneste er derfor definert ut fra de meldingsutvekslingsmønstrene den støtter. Et schema for dataene er inneholdt i meldingen og brukes som hoveddelen av kontrakten mellom bruker og leverandør. Andre typer metadata beskriver nettverksadressen for tjenesten, operasjoner den støtter, krav til pålitelighet, sikkerhet og transaksjonalitet.

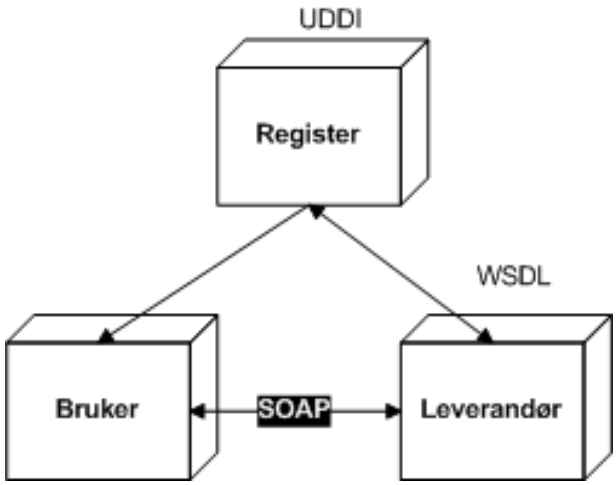
Interoperabilitet mellom to objekter, eller i dette tilfellet komponenter, kan forekomme på tre ulike måter (Chen & Doumeingts 2003);

1. ved at de to komponentene er integrert med hverandre
2. ved at komponentene stammer fra den samme metamodellen
3. ved at metamodellene for de to komponentene er tilpasset hverandre

Som nevnt har en tjeneste vanligvis et grovkornet grensesnitt som aksepterer mer data i én transaksjon enn et objekt, og som bruker flere maskinressurser enn et objekt fordi tjenesten må mappe innholdet i transaksjonen til en utførende enhet, prosessere XML og aksessere denne.

Tjenester er designet for å løse interoperabilitetsproblemer mellom applikasjoner og i utviklingen av nye applikasjoner, men ikke for å utvikle detaljert forretningsslogikk for applikasjoner. Tjenester utføres i stedet ved å utveksle meldinger i henhold til en eller flere utvekslingsmodeller, slik som query/reply (spørring/svar), enveis asynkron eller publish/subscribe (tilbud/abonnement). Muligheten for å koble gjenbrukbare tjenester sammen til større tjenester på en enkel måte er det som gir fordeler i form av

prosessautomatisering og rask respons på endringer i en tjenestes vilkår (Newcomer 2004). I en IT-organisasjon vil ansvarsområdene for hvordan designe, utvikle og ta i bruk applikasjonene som bruker tjenester kunne skilles mellom å opprette tjenesten og å bruke tjenesten. Denne ansvarsfordelingen skiller også tekniske spørsmål fra virksomhetsrelaterte spørsmål.



Figur 2.5: Web services arkitektur, hentet fra Newcomer (2004)

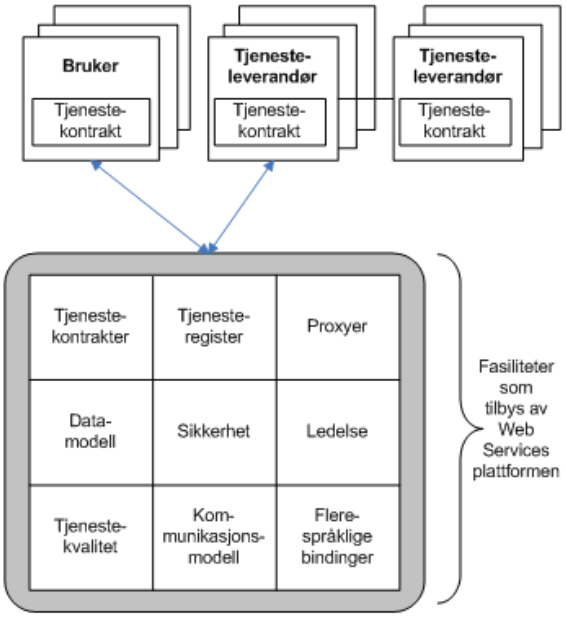
2.4.6 Tjenesteimplementasjon

Tjenesteimplementasjonen er et kjøringsmiljø som støtter Web services, og er ansvarlig for å implementere prosesseringsmodellen som er definert i de ulike tjenestespesifikasjonene. Kjøringsmiljøet er som regel et programvaresystem eller et programmeringsspråk. Et viktig moment i definisjonen av en tjeneste er at beskrivelsen er atskilt fra tjenesteimplementasjonen. Én beskrivelse kan ha flere implementasjoner, og én implementasjon kan støtte flere beskrivelser. Transformasjonslaget mellom beskrivelsen og omgivelsene er som oftest implementert ved hjelp av proxyer. Transformasjonslaget er ansvarlig for å akseptere meldinger, transformere XML-data til riktig format og klarere data til tjenesteimplementasjonen.

Web services inkluderer rollene bruker og leverandør. Brukeren tar initiativ til en tjeneste ved å sende en melding til en leverandør som på sin side utfører den tjenesten brukeren ba om. Det er imidlertid viktig å huske at en bruker kan også være en leverandør, og omvendt. En av de største fordelene med tjenesteabstraksjon er mulighetene for enkel aksess til et antall tjenestetyper.

Noe av det som er nytt med en tjenesteorientert arkitektur i forhold til andre designprinsipp, er muligheten til å bruke ulike kjøringsmiljø, å skille grensesnitt fra teknologien som utfører tjenestene, å la IT-avdelinger avgjøre hva som er beste omgivelse for hver jobb, og å kunne koble disse sammen ved å bruke en konsistent arkitektonisk tilnærming. Tidligere implementasjoner av tjenesteorientert arkitektur var basert på ett enkelt kjøringsmiljø og én enkelt teknologi, og muligheten til å skille mellom tje-

nestebeskrivelsen og kjøringsmiljøet er ny med tjenesteorientert arkitektur slik vi kjenner det i dag (Newcomer 2004). De løse koblingene gjør at ytelsesimplikasjonen er negativ, men i mange tilfeller er ytelse mindre viktig enn å enkelt kunne oppnå interoperabilitet. Skillet mellom tjenestebeskrivelsene og den teknologiske implementasjonen betyr at organisasjonen kan planlegge IT-investeringer rundt realiseringen av operasjonelle organisasjonsspørsmål, representert av beskrivelsene.



Figur 2.6: Web services plattformen, hentet fra Newcomer (2004)

2.4.7 Tjenesteintegrasjon

Forretningsintegrasjon i moderne tid har endret seg fra standarder som tillater heterogene systemer til viktigheten av standarder og systemer som samhandler effektivt. IBM ser tjenesteorientert arkitektur som en nøkkel til kravene om samhandling og fleksibilitet ifølge visjonen om organisasjoner som er "on demand" (Newcomer 2004). Tjenesteorientert arkitektur støtter end-to-end integrasjon på tvers av organisasjoner og mellom forretningspartnere. Når man bruker tjenesteorientert arkitektur for å designe distribuerte systemer kan man utvide bruken av Web services fra enkle klient-tjener modeller til systemer av vilkårlig kompleksitet.

En tjeneste i tjenesteorientert arkitektur er en applikasjonsfunksjon som er pakket som en gjenbrukbar komponent til bruk i en forretningsprosess, enten den tilbyr informasjon eller den gjør en endring i data fra én tilstand til en annen. Web services er basert på kall ved å bruke meldinger i en tjenesteorientert arkitektur som er beskrevet ved å bruke WSDL over en standard protokoll som HTTP.

Som jeg var inne på har organisasjoner de senere årene sett behovet for integrerte løsninger og produkter som er spesielt utviklet med tanke på dette. Slike produkter har imidlertid vist seg å være svært dyre, de krever stor innsats, implementasjonen tar uforholdsmessig lang tid, og de er i mange tilfeller en av årsakene til at prosjekter feiler (Newcomer 2004). Erfaring viser at en bedre løsning er tilgjengelig ved å bruke standarder fra Web services, fordi det er mulig å legge til åpne og abstraherte lag, de er basert på standarder og de er relativt enkle å integrere med både nye og eksisterende omgivelser. Det finnes i dag mange integrasjonsprodukter som har sitt utspring i tjenesteorientert arkitektur. Kombinasjonen av Web services og tjenesteorientert arkitektur tilbyr en raskere integrasjonsløsning som fokuserer på deling av data og gjenbrukbare tjenester i stedet for eiendomsrett til integrasjonsproduktene. Fordi grensesnittet til en Web service kan nås fra mange ulike klienter, er tjenesteorientert arkitektur med Web services godt egnet til å gi flere kanaler adgang til tjenestene som en gitt organisasjon ønsker å tilby.

2.4.7.1 Metadata i Web services

Styring av metadata inkluderer beskrivelser av en Web service som er nødvendig for å opprette en meldingskropp (message body) og et meldingshode (message header) slik at en tjenestebruker kan overvåke en tjeneste. For en bruker er det viktig å vite om datatyper og strukturen disse sendes på, men også andre kvalitetsattributter som sikkerhet, pålitelighet og transaksjoner. Spesifikasjoner av metadata inkluderer XML-Schema, WSDL, WS-Adressing, WS-Policy og WS-Metadata-Exchange. Kobling til tjenester gjøres ved først å finne tjenesten, noe som gjerne kan gjøres dynamisk. Dette står i kontrast til koblinger for tjenesteorientert arkitektur bygget på for eksempel J2EE, hvor alle koblinger må gjøres via pekere, referanser og lignende.

2.4.8 Sikkerhet og adressering

Adressering er et viktig krav til utvidede Web services fordi det ikke eksisterer noen katalog over endepunkter på Internett. I tillegg er poliser[3] nødvendige for at en tjenestebruker skal kunne tilby en passende sikkerhet, transaksjon og pålitelighet, og oppfylle datakravene for meldingene som er uttrykt i WSDL. Sikkerhet er viktig på alle nivå i en spesifikasjon av en Web service, og det kreves derfor ulike mekanismer for å beskytte seg mot utfordringer og trusler som er en del av distribuert prosessering (Newcomer 2004). Grunnleggende sikkerhetsbeskyttelse er bygget rundt kryptering, autentisering, autorisasjon og inkluderer gjerne logging for å kunne spore problemer som måtte oppstå.

Hver rolle i en interaksjon validerer sine poliser. En bruker må kunne stole på omgivelsene til en tjeneste, og en leverandør må kunne stole på omgivelsene til bruker. Når en bruker oppdager en tjeneste som ikke har en identitet kan oppdagelsen være anonym. Dette kan være en risiko dersom meldingsutvekslingen krever utveksling av sensitiv informasjon, som rapportering av individdata i Kostra. Dette utgjør en stor forskjell mellom manuell og automatisk oppdagelse av tjenester. For å unngå risikoen med ukjente tjenester kan man la en såkalt agent automatisk finne kandidater til tjenester og deretter la det være opp til en menneskelig bruker å avgjøre hvorvidt denne skal brukes. Sikker meldingsutveksling sikrer fortrolighet, konfidensialitet og integritet for interaksjonene og partene som utveksler informasjon.

Generelt sett kan sikre meldingsteknikker som kryptering og signaturer av meldinger brukes for routing og pålitelig meldingsutveksling. XML-baserte sikkerhetsteknologier er ofte viktige for å beskytte XML-data før og etter at det er inkludert i en SOAP-melding. Teknologiene inkluderer XML-Encryption og XML-Signatur. Disse kan brukes for å beskytte metadata så vel som data. Pålitelig meldingsutveksling er prinsippet som garanterer at en eller flere meldinger mottas et passende antall ganger. Spesifikasjon av dette inkluderer sikkerhetsmekanismer som WS-Reliability og WS-ReliableMessaging, og er designet for å sikre pålitelig levering av SOAP-meldinger over usikrede nettverk, slik som over Internett ved hjelp av HTTP. Det finnes også spesifikasjoner for varsling og publish/subscribe i WS-Eventing og WS-Notification.

Siden Kostra blant annet brukes til å utveksle personsensitive opplysninger over Internett, er sikkerhet et nøkkelord for en slik infrastruktur. Sertifiseringsteknologi, digital signatur og PKI som er integrert i en tjenesteorienterte arkitekturen, støtter alle autentisering av brukerne av en tjeneste. Man har også i det siste standardisert bruk av digital signatur og kryptering for XML-dokumenter, XML-Dsig og XML-Enc, som av mange regnes som den beste måten å bruke PKI på i en tjenesteorientert arkitektur (Baglietto et al. 2004)

[3]Poliser er maskinlesbare uttrykk av regler som en bruker av en tjeneste må følge for å kunne kontakte en leverandør

2.5 Fordeler og ulemper med en tjenesteorientert arkitektur

I korte trekk springer fordelene ved bruk av tjenesteorientert arkitektur ut fra lavere applikasjonsutviklingskostnader, lavere vedlikeholdskostnader, økt endringsdyktighet i organisasjonen samt økt pålitelighet ved å kunne designe systemer som er mer resistente i forhold til feil og avbrudd. I tillegg er oppgradering av applikasjoner betydelig enklere og krever mindre avbrudd enn total applikasjonsutskiftning som hittil har vært vanlig i utviklingsmiljøer.

2.5.1 Abstraksjon

Tjenesteorientert arkitektur med Web services har vist at det kan tilby riktig abstraksjonsnivå for å kunne ta seg av ulik teknologi og løse koblinger, som på sin side er nødvendig for å kunne gjøre gjenbruk av virksomhetstjenester for applikasjoner til flere kanaler. Det er flere fordeler knyttet til en overgang fra enhetlige applikasjoner til systemer basert på flere kanaler ved hjelp av tjenesteorientert arkitektur. Utvikling fra enhetlige applikasjoner til flerkanalssystemer basert på tjenesteorientert arkitektur åpner for muligheter som å splitte og slå sammen applikasjoner, som på sin side reduserer kostnader og øker organisasjonens effektivitet. De største arkitektoniske utfordringene med flerbrukeraksess er balansen mellom ulikt sluttbrukerutstyr og de ulike teknologiene man planlegger å ta i bruk. Disse teknologiene har blant annet kjennetegn som tilkoblingsmuligheter, sikkerhet, kommunikasjonsteknologi, som alle vil variere fra teknologi til teknologi.

2.5.2 Integrasjon

Tjenesteorientert arkitektur er introdusert for å kunne gjøre integrasjon på et høyere nivå enn for eksempel arkitekturer med distribuerte objekter (DOA), og tjenesteorientert arkitektur på derfor være med teknologinøytral og dekke en større del av organisasjonen, og også kunne integrere systemer på tvers av organisasjoner (Baker & Dobson 2005). Man kan diskutere om tjenesteorienterte arkitekturer en en tilpasning av DOA, om det er en ny type mellomvare eller en forretningsdrevet erstatning for gammel teknologi. På modelleringsnivå har DOA en tilnærming som er klart objektorientert, fordi man identifiserer hver abstraksjon og gjør denne tilgjengelig som et eget objekt. Den vanligste sammenlikningen mellom DOA og tjenesteorienterte arkitekturer er at mens en tjenesteorientert arkitektur er teknologinøytral er DOA mindre fleksibel. Både tjenesteorienterte arkitekturer og DOA er likevel strukturerte rundt tjenester på den ene siden og objekter på den andre siden, som begge utfører gitte handlinger på vegne av en klient (Baker & Dobson 2005). I en tjenesteorientert arkitektur vil man derimot dekomponere virkeligheten i form av reelle entiteter som har faktisk eksistens på forretningsnivå, og som fanger opp interaksjoner mellom forretningsentitetene.

2.5.3 Gjenbrukbarhet

Den tjenesteorienterte arkitekturen og dens komponenter er i stor grad gjenbrukbare. Stor grad av gjenbrukbarhet er helt klart en fordel for en organisasjon, fordi gjenbrukbarhet gjør at deling på tvers av avdelinger blir enklere, og kostnadene ved utvikling og styring av nye komponenter, som i stor grad består av tidligere utviklede tjenester, blir lavere. Tjenestene i en tjenesteorientert arkitektur er abstrahert fra implementasjonen av deres funksjonalitet, noe som fjerner avhengigheter tjenestene ellers ville hatt på den underliggende implementasjonen.

2.5.4 Formelle metoder

I likhet med andre arkitekturer bygger den tjenesteorienterte arkitekturen på formelle metoder. De formelle kontraktene gjør at kvaliteten på grensesnittene er høyere og at de er forstått av begge parter. Tjenesteorienterte arkitekturer har likevel ikke standardiserte grensesnitt, i motsetning til DOA. Fordelen med dette er at det øker fleksibiliteten og dynamikken ved utvikling, og tjenesteorienterte arkitekturer står på den måten bedre rustet enn DOA ved integrasjon av flere ulike systemer. Dette skjer til gjengjeld på bekostning av økte ressurser brukt på tilpasning og vedlikehold av så mange og ulike grensesnitt.

2.5.5 Grensesnitt

Fokuset på grovkornede grensesnitt i en tjenesteorientert arkitektur kan likevel være med på å forenkle interaksjon på tvers av organisasjoner ved å redusere det antallet grensesnitt man må igjennom for å bli forstått av en annen tjeneste eller et annet system. Dette er likevel en viktig forenkling i en verden hvor organisasjoner har en mer dynamisk interaksjon enn tidligere. Det faktum at tjenesteorienterte arkitekturer ikke vektlegger grensesnittyper vil sannsynligvis ikke føre til enklere integrasjon på lengre sikt. Selv om det blir enklere å etablere nye grensesnitt, vil slike ad-hoc-tilnærminger føre til større kompleksitet og kostnadene ved videre utvikling vil med stor sannsynlighet øke (Baker & Dobson 2005).

2.5.6 Innføring av en tjenesteorientert arkitektur

I og med at tjenestene skal publiseres og nettverket i en viss grad åpnes, kreves det at organisasjonen har tilstrekkelig sikkerhetsinfrastruktur for å hindre eventuelle angrep utenfra. Rent organisatorisk krever innføring av tjenesteorientert arkitektur at aktuelle medarbeidere i organisasjonen får nødvendig opplæring og ferdigheter for å kunne ta arkitekturen i bruk, i tillegg til at innføringen av en ny arkitektur krever at nye roller bekles av de ansatte i organisasjonen. Utviklingsmessig krever innføringen av tjenesteorientert arkitektur at man sikrer at arkitekturen kan anvendes på hele organisasjonen. Arkitekturen krever videre at tjenestene har tilstrekkelig abstraksjon fra implementasjonen av funksjonaliteten, og at brukere og leverandører av de samme tjenestene samarbeider under hele

utviklingsprosessen. Eksisterende systemer må også kunne pakkes inn i tjenester.

2.5.7 Investeringer og avkastning

Organisasjonsintegrasjon i moderne tid har endret seg betraktelig fra standarder som tillater heterogene systemer til effektivt samhandlende systemer, og blant annet ser IBM tjenesteorientert arkitektur som en nøkkel til kravene om samhandling og fleksibilitet jamfør visjonen om fleksible og endringsdyktige organisasjoner.

Den reelle verdien av tjenesteorientert arkitektur kommer når nye applikasjoner i stor grad kan utvikles ved å bruke eksisterende tjenester. Det tar imidlertid tid å komme opp på et slikt nivå, og det krever betydelige investeringer innen tjenesteutvikling. Nøkkelen til en suksessfull arkitektur er å finne korrekt design og funksjon for alle tjenestene i biblioteket over gjenbrukbare tjenester. Dette biblioteket bør også reflektere forretningsprosessene i organisasjonen. Det er disse prosessene som skal automatiseres, og et vellykket tjenesteorientert prosjekt vil sørge for at gjenbrukbare tjenester tilnærmes forretningsprosessene, som igjen sikrer at prosessene kan endres i henhold til endringer i omgivelsene. Utfordringene ved en tjenesteorientert tilnærming ligger likevel i kursing av ansatte og å sørge for at tjenestene som utvikles er gjenbrukbare. Tjenester må utvikles med tanke på langsiktighet, og ikke bare med umiddelbare fordeler. En tjeneste har ingen stor verdi dersom den ikke kan brukes av flere applikasjoner og flere nye applikasjoner kan utvikles med bakgrunn i denne tjenesten. En annen utfordring er de kortsiktige kostnadene knyttet til tjenesteorientert utvikling. Å bygge opp en tjenesteorientert arkitektur er ikke en billig investering, men avkastningene blir større over tid. Dette krever at analytikerne definerer forretningsprosessene, systemarkitektene må gjøre prosesser om til spesifikasjoner, systemutviklerne må utvikle ny kode og prosjektlederne må styre det hele. Noen spesifikasjoner må også endres for å kunne passe inn i en tjenesteorientert arkitektur.

Kapittel 3

Arkitekturmodell

En god og fornuftig IT-arkitektur er en avgjørende faktor dersom offentlig sektor skal gjennomføre sine visjoner om digital forvaltning. Formålet med felles arkitekturprinsipp i offentlig sektor er å sikre bedre sammenheng i offentlig forvaltning, både internt, og overfor borgere og næringsliv hvor det ofte handler om samarbeid og samhandling på tvers av etater. IT-arkitektur er ett av leddene i en lang prosess som skal føre til bedre service, bedre tjenesteyting og en mer effektiv forvaltning av disse tjenestene. Ministeriet for Videnskab Teknologi og Udvikling i Danmark har laget en rapport kalt "Grønbog om IT-arkitektur" (IT og Telestyrelsen 2002) hvor de beskriver betydningen av en velfungerende IT-arkitektur i offentlig sektor.

3.1 IT-arkitektur

Definisjoner av IT-arkitektur:

> Software architecture is the set of design decisions, if made incorrectly, may cause your project to be cancelled
>
> — *Eoin Woods (Carnegie Mellon University 2006)*

> The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evloution over time
>
> — *David Garlan (Carnegie Mellon University 2006)*

> Den fundamentale organisering af et system, indbygget i dets komponenter, deres innbyrdes relationer og i de prinsipper, som styrer dets design og udvikling.
>
> — *IT og Telestyrelsen (2002)*

De viktigste fordelene ved en tjenesteorientert arkitektur er interoperabilitet, som gjort riktig fører til at data og programvaremoduler kan utveksles og brukes i flere sammenhenger. Andre fordeler er at deler av tjenestene

kan settes i drift relativt raskt, på tross av at hele systemet ikke er ferdig utviklet, tjenester kan være tilgjengelige døgnet rundt og man kan bygge inn redundans på alle nivåer i arkitekturen. Data og funksjoner kan gjenbrukes og det blir enklere å bruke eksisterende produkter på markedet og å samarbeide med eksterne aktører. Skalerbarheten fører til etterspørselsdrevet utvikling og drift, og støtte for flere medier åpner for integrasjon av alle typer informasjon, det være seg bilder, lyd og video via samme IT-infrastruktur (IT og Telestyrelsen 2002).

IT-arkitektur kan være nøkkelen til å skape effektive og sammenhengende løsninger, fordi dårlig IT-arkitektur ofte er årsaken til det motsatte. Behovet for å utvikle en spesifikk IT-arkitektur kan derfor begrunnes med at det gir et teknisk grunnlag for en effektiv og sammenhengende IT-strategi.

En IT-arkitektur skal gi en strategisk ramme og kontekst for utviklingen av IT-løsninger. En god IT-arkitektur gjør det lettere å skape en god balanse mellom forretnings- og forvaltningsmessig innovasjon på den ene siden og IT-effektivitet på den andre siden. IT-arkitektur har betydning både i forhold til design av det enkelte system, og i forhold til sammenheng og kommunikasjon mellom mange ulike systemer. Formålet med en felles IT-arkitektur er å forbedre mulighetene og sammenhengen mellom brukerne av IT-systemer. Ved å begrense den enkeltes handlefrihet er det mulig å optimere fellesskapets interesser (IT og Telestyrelsen 2002).

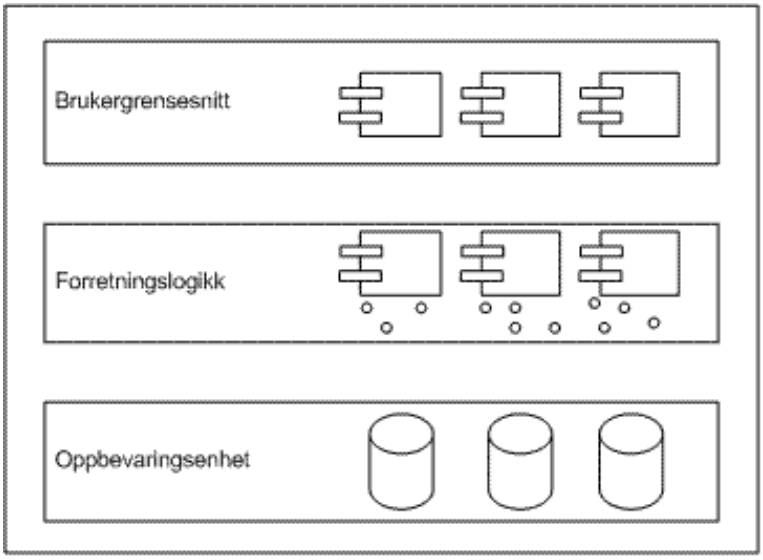
Dersom formålet er å skape sammenheng og effektivisering på tvers av avdelinger og aktører bør det altså legges stor vekt på arkitektoniske valg som støtter interoperabilitet og fleksibilitet. Offentlig sektor har en stor utfordring i å skulle fastholde kvaliteten i de offentlige tjenestene. Man bør derfor bygge digitale løsninger som kan skape en mer effektiv forvaltning, og som enten fastholder eller forbedrer nivået for offentlige tjenester, og hvor man oppnår en mer effektiv oppgaveløsning som gagner borgere så vel som organisasjoner (IT og Telestyrelsen 2002).

Den teknologiske utviklingen går for alvor i retning av portaler som en ramme for digital forvaltning (e-eGovernment) og digital forretning (e-business). Portalene bygges i økende grad av komponenter som er designet for å kunne integrere ulike systemer og funksjoner, som igjen skal kunne integreres hurtig og enkelt i ulike portaler.

3.1.1 Flerlagsarkitektur

Flerlagsarkitekturen er et uttrykk for en logisk modell for hvordan et IT-system kan forstås som en rekke moduler som hver har veldefinerte grensesnitt, se figur 3.1 på neste side. Brukergrensesnittet gir brukeren adgang til gitte tjenester gjennom ulike medier, forretningslogikken er applikasjonslaget som holder logikken som utfører tjenestenes funksjonalitet og oppbevaringslaget inneholder de faktiske data og består av systemer som skaffer og vedlikeholder elektronisk info og dets metadatabeskrivelser.

Den lagdelte arkitekturmodellen er en logisk modell, som må suppleres med en mer normativ modell for systemarkitekturens designkrav og implementasjonsstrategi. En mer normativ modell kalles gjerne en tjenesteorientert arkitektur. Fordelene med en slik lagdeling er at utviklingen kan



Figur 3.1: Flerlagsarkitektur, hentet fra IT og Telestyrelsen (2002)

gå raskt fordi man kan utvikle de ulike lagene i parallell, tilgjengeligheten for hvert enkelt lag blir god, man kan dra nytte av gjenbruk av funksjoner og eksisterende produkter på markedet, skalerbarheten blir god og man kan integrere alle typer informasjon via samme IT-arkitektur. Ulempene eller barrierene med en slik arkitekturmodell er at initialkostnadene til utvikling og innkjøp av komponenter og moduler gjerne blir høye. Det kreves også en god tilpasning av nye løsninger til de eksisterende løsningene i organisasjonen, og man må integrere og tilpasse eksisterende fagsystemer.

Tjenesteorientert arkitektur er en modell for distribuert databehandling i dette tilfellet en modell for offentlige e-tjenester og interne tjenester mellom systemer i én og samme organisasjon (IT og Telestyrelsen 2002). Tjenester kan i denne sammenhengen være både modeller, data og funksjoner, en gruppe standarder eller applikasjoner.

En gradvis innføring av en tjenesteorientert arkitektur i digital forvaltning kan for eksempel se slik ut (IT og Telestyrelsen 2002):

1. organisk innføring av verktøy og standarder

2. systematisk spredning og anvendelse av tjenesteinfrastrukturen

3. gjennomgående bruk av tjenester i tjenestefellesskap

Det er imidlertid viktig å være klar over at dette tar tid, spesielt fordi det gjerne krever endringer i forretningsprosesser i organisasjonen. Informasjonssystemer generelt stiller også krav til mange ulike typer kompetanse. Bare med tanke på sikkerhet er det viktig med tillit til et system. Åpenhet og fortrolighet er andre viktige elementer, samt at man tar hensyn til lover om persondata, noe som gjerne håndheves gjennom blant annet kryptering, digital signatur, sertifiserte servere og adgangskontroller.

Finansiering er ofte fremhevet som det største problemet med nye arkitekturer. IT er ikke bare en driftsfunksjon, men kan ha avgjørende betydning for utvikling av forretningsprosesser og styring. Det er her viktig å avklare spørsmål som hvem i organisasjonen som sitter på ansvaret for å etablere en IT-arkitektur, og å sørge for at en utvikling av denne er i overensstemmelse med retningslinjer, forretningslogikk osv i organisasjonen.

3.2 Applikasjonsintegrasjon

Applikasjonsintegrasjon er en strategisk tilnærming for å knytte flere informasjonssystemer sammen for å utnytte og utvikle deres muligheter for dermed å kunne utveksle informasjon og utføre prosesser i sann tid. Applikasjonsintegrasjon kan ha mange former, inkludert integrasjon av interne applikasjoner som Enterprise Application Integration eller integrasjon av eksterne applikasjoner som Business-to-Business Application Integration. Selv om disse to hovedformene for applikasjonsintegrasjon hver har sine spesialiteter har de interne og eksterne integrasjonsløsningene fortsatt mange likheter.

3.2.1 Ulike former for applikasjonsintegrasjon

En klar trend innen system- og applikasjonsintegrasjon er at man stadig flytter seg vekk fra informasjonsorientert integrasjon til tjenestebasert integrasjon (Linthicum 2004). Informasjonsorientert integrasjon er en relativt billig måte å integrere applikasjoner på, fordi man i de fleste tilfeller ikke trenger å endre på selve applikasjonen. Selv om informasjonsorientert integrasjon gir en funksjonell løsning på mange problemer, er det integrasjon av både tjenester og metoder som gir best avkastning på lang sikt.

Selv om tilnærmingene til applikasjonsintegrasjon varierer, har Linthicum (2004) likevel valgt å dele disse inn i fire kategorier; informasjonsorientert applikasjonsintegrasjon (IOAI etter Information Oriented Application Integration), integrasjon av forretningsprosesser (BPIOAI etter Business Process Integration-Oriented Application Integration), tjenesteorientert integrasjon (SOAI etter Service Oriented Application Integration) og til slutt portalorientert applikasjonsintegrasjon (POAI etter Portal Oriented Application Integration).

Generelt kan applikasjonsintegrasjon gi fordeler til de fleste industrier, blant annet gjennom å kunne imøtekomme kunder raskere, og å kunne gjennomføre forretninger raskere. Det er likevel viktig å huske at det viktigste ikke er hva man gjør men hvordan man gjør det. Applikasjonsintegrasjon er av liten verdi dersom det ikke raskt kan tas i bruk, hvis det ikke er korrekt implementert eller hvis det ikke kan endres i takt med endringer i organisasjonens omgivelser. Applikasjonsintegrasjon er i det store og hele kun en strategisk anvendelse av teknologi for å tilby organisasjoner den infrastrukturen de trenger for å kunne håndtere de fleste hendelser elektronisk og i sann tid (Linthicum 2004).

3.2.1.1 Information oriented Application Integration, IOAI

Det at IOAI er forholdsvis ukomplisert og at det tar relativt kort tid å implementere en slik integrasjon, er en konsekvens av at forretningslogikken sjelden eller aldri må endres (Linthicum 2004). De fleste brukere vil derfor ikke merke at data deles i bakgrunnen, fordi oppførselen til applikasjonene ikke endres. Man antar at systemene som inngår i problemområdet både kan bruke og produsere informasjon, og man kan dermed kategorisere de ulike informasjonskildene slik:

- database
- applikasjon
- brukergrensesnitt
- tilleggsutstyr

Databaser er gjerne et naturlig integrasjonspunkt, nettopp fordi de er designet for å produsere og bruke data. Databaser har dermed det beste grensesnittet for å utveksle informasjon med applikasjoner. Den klare fordelen med databaser utover dette er at grensesnittene er godt definerte og gjennomtenkte, samt at man kan be om mange ulike typer resultat. Ulempen er naturligvis at informasjonen som produseres av en database vanligvis ikke er bundet til noen instans av forretningen som databasen er en del av (Linthicum 2004). Man må derfor, for hvert resultat, finne ut av hvordan man skal bruke nettopp denne informasjonen.

Applikasjonsgrensesnitt er mer komplekse enn grensesnitt mot et medium som en database fordi applikasjoner har en annen tilnærming til hvordan de bruker og produserer informasjon, hvis de overhodet gjør dette. Brukergrensesnitt er bare én av teknikkene og teknologiene som kan brukes for å aksessere, eller plassere informasjon i en applikasjon. Brukergrensesnitt har vært i bruk gjennom flere år, så risikoen knyttet til bruken av disse vil ikke være særlig høy. Det gjenstår imidlertid å nevne at de første brukergrensesnittene aldri ble designet for å skulle utveksle data. Dette skaper blant annet problemer i forhold til skalering, da denne typen løsning ikke kan skaleres og at det dermed er umulig å håndtere mer enn noen få skjermbilder samtidig. Det er naturligvis flere måter å unngå disse begrensningene på, men man må da huske at man også tilfører prosjektet økte risiki (Linthicum 2004).

3.2.1.2 Business Process Integration-Oriented Application Integration, BPIOAI

Gjennom BPIOAI definerer man en felles forretningsprosessmodell som adresserer sekvens, hierarki, hendelser, logikk og informasjonsflyt mellom systemer (Linthicum 2004). Ideen med BPIOAI er å skulle tilby én logisk modell som spenner over flere applikasjoner og datalagre og som tilbyr en notasjon for felles forretningsprosesser som kontrollerer hvordan systemer og mennesker samhandler for å kunne utvikle unike forretningskrav. Målet

er å abstrahere applikasjonstjenestene og applikasjonsinformasjonen i én enkelt kontrollerende forretningsprosessmodell (Linthicum 2004).

Å integrere applikasjoner i form av BPIOAI-løsninger krever at man fjerner de eksisterende avhengighetene mellom applikasjonene. Routing-løsningen i mange BPIOAI-løsninger tillater at informasjon hentes ut fra en hvilken som helst kildeapplikasjon, målapplikasjon eller datalager. Løsningene tillater også kall på applikasjonstjenester. Fordelen med BPIOAI er at det kun er modellen i seg selv som må endres når man først må endre en prosessflyt eller logikk, og man trenger ikke endre applikasjoner som er en del av prosessmodellen. Dette gjør at man kan gjenbruke ethvert kilde- eller målsystem fra modell til modell (Linthicum 2004). Det viktige når man velger å implementere BPIOAI er at forretningsprosesser i organisasjonen og hos en eventuell samarbeidspartner må dokumenteres, de prosessene som mangler må defineres og prosessen som bruker BPIOAI-teknologi for å knytte disse prosessene sammen må utføres.

3.2.1.3 Service Oriented Application Integration, SOAI

Målet med denne integrasjonsteknikken er å kunne få tilgang til applikasjonstjenester fra fjerntliggende applikasjoner gjennom Internett, et godt definert grensesnitt og registertjenester.

Tjenesteorientert applikasjonsintegrasjon lar applikasjoner dele en felles forretningslogikk og noen felles metoder. Dette gjøres enten ved at de aktuelle metodene deles, og derfor integreres, eller ved å tilby infrastruktur for slike metoder, for eksempel Web Services. Gjenbruk er også et verdifullt mål. Et felles sett av metoder som dekker flere applikasjoner inviterer til gjenbruk, noe som igjen er med på å redusere behovet for overflødige metoder og/eller applikasjoner. Den største ulempen med denne integrasjonstilnærmingen er, som nevnt tidligere, økte kostnader som blant annet kan knyttes til endringer i applikasjoner og metoder, i motsetning til en tilnærming med informasjonsorientert integrasjon. Tjenesteorientert applikasjonsintegrasjon er likevel best egnet i mange problemområder (Linthicum 2004), man bør bare vokte seg slik at man kun bruker tjenesteorientert arkitektur på de områdene man virkelig har behov for det.

SOAI lar organisasjoner dele felles applikasjonstjenester på lik linje med informasjon (Linthicum 2004). Gjenbruk er etter hvert blitt et viktig begrep, og et sett med applikasjonstjenester blant virksomhetssjonsapplikasjoner inviterer til gjenbruk og reduserer dermed behovet for overflødige tjenester og/eller applikasjoner. Som nevnt krever IOAI få eller ingen endringer i eksisterende mål- eller kildeapplikasjoner. SOAI derimot krever en endring i de fleste, om ikke alle, virksomhets og B2B[1]-applikasjoner for å få et størst mulig utbytte av endringsmønstret. Dette er den største ulempen med SOAI, og er en ulempe som gjør integrasjonsteknikken vanskelig å selge inn i organisasjoner.

[1] business-to-business

3.2.1.4 Portal Oriented Application Integration, POAI

POAI tillater å vise flere systemer, både interne og eksterne, gjennom ett enkelt grensesnitt eller én enkelt applikasjon. POAI unngår både integrasjonsproblemene ved å forlenge brukergrensesnittet for hvert system til et felles brukergrensesnitt, i de fleste tilfeller en nettleser. POAI integrerer derfor alle deltagende systemer gjennom en nettleser, selv om dette egentlig ikke integrerer applikasjonene i eller mellom organisasjoner (Linthicum 2004).

Det er viktig å være klar over at portaler er blitt den primære integrasjonsmekanismen og bruk av portaler for å integrere virksomheter har mange fordeler. Den mest fremtredende er at man ikke behøver integrere back-end systemene direkte, verken innenfor en gitt organisasjon, eller mellom bedrifter. Andre fordeler er at POAI lar andre organisasjoner samhandle med et selskaps interne systemer gjennom et regulert grensesnitt som er tilgjengelig via Internett. I tillegg er en slik integrasjonsløsning raskere å implementere enn sann tids utveksling med back-end systemer (Linthicum 2004).

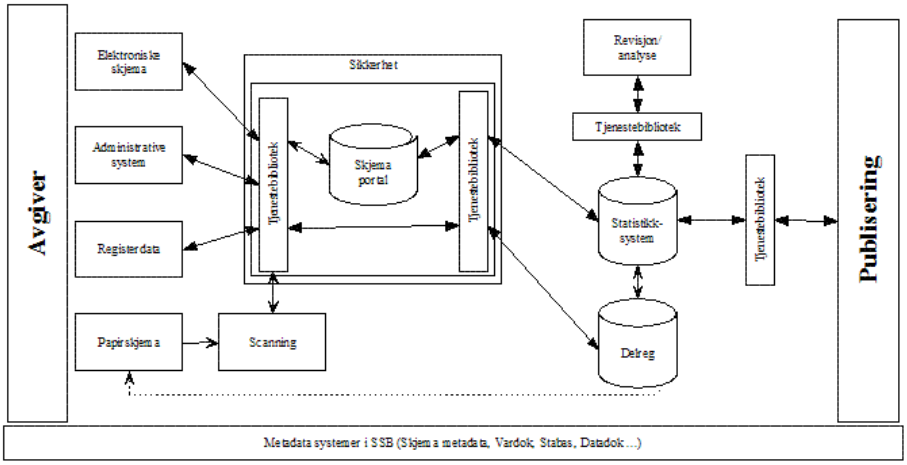
En av ulempene knyttet til POAI er at informasjonen ikke strømmer i sann tid, og dermed krever brukerinteraksjon. Et annet problem er at informasjon må spres, typisk gjennom andre applikasjonslag, noe som er med på å gjøre systemløsningen mer kompleks enn hva som var utgangspunktet. Til slutt er det viktig å nevne at sikkerhet alltid vil være et viktig tema for informasjon og data som transporteres og utveksles over Internett. Dessverre støtter ikke portaler det ultimate målet med applikasjonsintegrasjon, fordi portaler ikke utveksler forretningsinformasjon i sann tid og samtidig krever stor grad av interaksjon med en bruker. På tross at dette er portaler like fremt en nødvendighet per i dag.

3.3 Rapporteringsløsninger i SSB

Kostra og Idun er de største rapporteringssystemene i SSB, og er utviklet henholdsvis for rapportering fra kommuner, fylkeskommuner og offentlige institusjoner, og for rapportering fra næringslivet generelt. I dette kapittelet følger en beskrivelse av disse to systemene og hvordan disse er realisert. Dokumentasjonen av Idun og Kostra (se hhv. kapittel 3.3.1 og 3.3.2) er i stor grad enten utdatert eller mangelfull. Beskrivelsen av systemene slik de foreligger i dag, er av den grunn i første rekke basert på intervjuer og samtaler med systemutviklere ved SSB, som arbeider eller har arbeidet med disse to systemene.

Gjennom en felles, online rapporteringsløsning for Kostra og Idun, vil skjemautfylling og datamottak kunne skje nærmest samtidig. Data fra avgiver lagres i SSBs datafangstportal, og kan for eksempel overføres til de aktuelle fagseksjonene én gang i døgnet ved hjelp av batch-overføringer. Figur 3.2 på neste side illustrerer en slik datafangstmodell hvor data fra ulike kilder transporteres gjennom flere felles kanaler i SSB. Disse kildene kan være data som hentes inn via portalen for elektroniske skjema, data fra AltInn, data fra ulike registre eller data som samles inn via papirskjema. Modellen har komponenten “Metadata” som en egen komponent, fordi

det p.t. er flere initiativ og prosjekter som arbeider med å kartlegge behov, omfang og løsninger i forhold til bruk av metadata i datafangstarbeidet (Sand, Holm, Pedersen, Moaffi, Dale, Folkedal, Hole & Krogsrud 2006). Det er viktig å være klar over at disse initiativene og prosjektene vil kunne påvirke de tekniske løsningene for en ny og felles datafangstmodell for Kostra og Idun.



Figur 3.2: Datafangstmodell, hentet fra Sand et al. (2006)

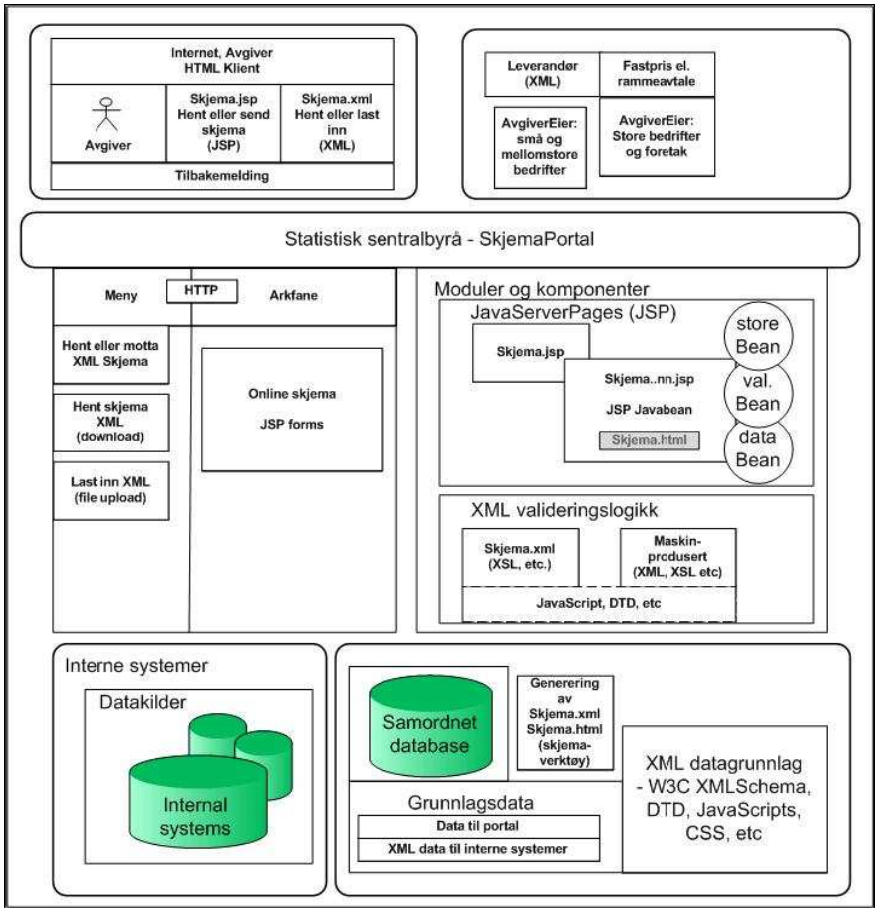
3.3.1 Idun

3.3.1.1 Systemets oppbygning

Idun er utviklet som en online portalløsning, og arbeidet med dette systemet startet i 1998. Portalen har i første rekke som formål å sette sammen, presentere og motta informasjon fra en avgiver av data til SSB, og deretter videresende avgivers forespørsel eller respons. Portalen er vertikalt strukturert, slik at nye moduler enkelt kan legges til ved behov. Det er naturlig nok ulike fordeler og ulemper knyttet til en slik portal, men valget ble i sin tid tatt på bakgrunn av økonomiske rammebetingelser, og med hensyn til de funksjonelle kravene i prosjektet. Det er samtidig viktig å merke seg at portalen for SSB ikke er et kommersielt produkt, men utviklet internt av de ulike IT-seksjonene ved SSB.

Idun som system er bygget opp av ulike moduler. Sikkerhetsmodulen i Idun har som formål i kontrollere at en avgiver av data har tilgang til systemet. Dette gjøres ved å sjekke hvem brukeren er gjennom pålogging, og for å kunne rapportere tall krever Idun pålogging med brukernavn og passord. De andre modulene i systemet vil deretter kontrollere om brukernavn og passord er gyldig for en avgiver for gitte operasjoner i systemet. Dette gjelder også for tilgangen til de ulike skjemaene i Idun.

Profilmodulen skal innhente, endre eller lagre informasjon og innstillinger i henhold til en gitt avgiver. Hver bruker har dermed sin profil i Idun, hvor innholdet i profilen er avhengig av innstillinger for og informasjon om denne brukeren.



Figur 3.3: Idun systemarkitektur, hentet fra Statistisk sentralbyrå (n.d.*a*)

Skjemamodulen har som oppgave å hente skjema, samt motta informasjon fra avgiver, og gi tilbakemeldinger på eventuelle feil som utføres i prosessen. Deretter skal skjemamodulen lagre dataene på en hensiktsmessig måte i henhold til databasens metadatastruktur.

Den siste modulen i Idun er rapportmodulen, som har som formål å gi avgiver tilgang på personifiserte rapporter, og eventuelt andre rapporter som faller inn under organisasjonens interesse eller næringsområde. Rapportene er delt inn i tre ulike kategorier; dynamiske rapporter, forhåndsgenererte rapporter og direkte generering.

I tillegg til disse modulene består Idun av et sett med valideringer og forretningslogikk. Valideringene er enten applikasjonskontroller eller overordnet validering av blant annet korrekte datatyper og null-verdier. Forretningslogikken består av både generell forretningslogikk som kan være gjeldende for flere skjema, eller det kan være skjemaspesifikk forretningslogikk, som kontroll og revisjon mot historiske data.

For at en gitt avgiver skal kunne rapportere sine tall til Idun og SSB, må han/hun som tidligere nevnt logge seg på systemet med et brukernavn og passord. I de fleste tilfeller kommer både brukernavnet og passordet sammen med papirskjemaet som brukeren mottar per brev. Dersom bedriften eller foretaket har avgitt data gjennom Idun tidligere skal disse imidlertid beholde og bruke det passordet de fikk tildelt ved første gangs rapportering til SSB via Idun. Idun mangler imidlertid gode rutiner for passord ved eierskifte i bedrifter, fordi passordet ikke er knyttet opp mot personen men mot enheten som logger seg på systemet. Når en bruker er logget inn i portalen får han/hun en oversikt over de skjema som er tilgjengelige for rapportering for denne bedriften/foretaket. Til hvert skjema er det i tillegg hjelpefunksjoner i form av veiledninger, og i noen tilfeller hjelpetekster til spørsmålene i skjemaet.

3.3.1.2 Skjemagenerering

Metadatabasen er selve kjernen i Idun, og er en Oracle-database med nær sagt identisk oppbygning som metadatabasen som eksisterer for Kostra. I den forbindelse har man nå påbegynt arbeidet med å opprette en felles metadatabase for disse to systemene. Databasen er inndelt i fire logiske deler som hver for seg representerer de ulike oppgavene som utføres av systemet. Meta-/faktadelen henter og lagrer innrapporterte data fra en avgiver, profildelen henter og lagrer informasjon om en avgiver, delen for skjema og validering henter, lagrer og validerer skjema, og portal-delen henter stiler/templates for å håndtere brukergrensesnittet mot avgiveren. I tilknytning til denne metadatabasen har SSB utviklet en applikasjon, SMETA, hvor skjema administreres og genereres på bakgrunn av informasjonen i metadatabasen.

Det er seksjon for IT fellestjenester ved SSB som har ansvaret for å generere skjemaene. Før arbeidet med å generere et nytt Idun-skjema starter avholdes det et møte med den fagseksjonen ved SSB, som har oppdragsgiverrollen overfor seksjon for IT fellestjenester. Under dette møtet avklarer man ansvarsområder og tidsfrister som skal knyttes til det

aktuelle skjemaet. Deretter er det oppdragsgivers oppgave å utarbeide en kravspesifikasjon for skjemaet, som blant annet skal inneholde tekstene i skjemaet, ønsker om automatiske valideringsmekanismer med mer. Når oppdragsgiver er ferdige med dette overlates kravspesifikasjonen til seksjon for IT-fellestjenester, som benytter SMETA for å generere skjemaet etter oppdragsgivers ønske. Når all metadata er lagt inn i SMETA genereres et utkast til skjema som legges inn i versjonshåndteringssystemet CVS. Dette må gjøres eksplisitt, fordi SMETA ikke har tilfredsstillende versjonshåndtering. Det må likevel legges til at applikasjonen er bedre med tanke på dynamikk sammenliknet med brukerapplikasjonen i Kostra, og SMETA har samtidig bedre visuell informasjon. Når skjemaet er lastet inn i CVS benytter skjemautviklerne en applikasjon for å generere det ferdige skjemaet slik det skal se ut for brukerne gjennom Idun-portalen. Applikasjonen plukker ut elementene som skal inngå i XML-filen som skjemaet er bygget opp av, et script bygger opp skjemaet ved å selektere informasjon om skjema, blokk, spørsmål, kolonner, rader og til slutt celler med tilhørende attributter. På bakgrunn av dette genereres en XML-fil. XML-filen kobles deretter sammen med en XSL-fil, og en java-applikasjon genererer HTML og JSP på bakgrunn av disse filene. Skjemaet gjøres deretter tilgjengelige slik at de kan testes av oppdragsgiver etter at testbrukere er koblet til skjema. Når oppdragsgiver har testet skjemaet gis det en tilbakemelding til seksjon for IT-fellestjenester med eventuelle korreksjoner som må gjøres i skjemaet.

Et kjent problem i Idun er mangelen på gode testdata. I mange tilfeller skal data preprintes i skjema, dvs. enkelte data fra tidligere år legges inn i skjema som skal fylles ut for å lette oppgavebyrden for næringslivet. Man er derfor helt avhengig av en reell bruker, noe som er vanskelig å oppfylle ved testing.

Når skjemaet er testet ferdig av fagseksjonen går skjemaet over i en produksjonsfase. Bedrifter og foretak knyttes til skjemaet, og skjemaet sammen med JSP-filer og veiledninger pakkes sammen. Skjemaet blir deretter preprintet i de tilfellene hvor dette er aktuelt, og skjemaet skal være klart senest tre dager før det skal være tilgjengelig for utfylling. Når skjemaet omsider er klart, sendes et brev fra SSB til alle avgivere med generell informasjon og en papirutgave av skjemaet, med unntak av tilfellene hvor skjema kun er i elektronisk format. Hver natt hentes skjemadata og kvitteringsinformasjon fra Idun-portalen. Filene lastes så over på en UNIX-server som er spesifisert i skjema av oppdragsgiver. Herfra er det oppdragsgiver, dvs fagseksjonene, som har ansvaret for skjemaet og for å overføre data til Statistikkbanken. Her kan brukerne av statistikken selv sette sammen data slik at det passer deres spesielle behov. Dette gjelder også for Kostra.

3.3.2 Kostra

3.3.2.1 Skjemaproduksjon

Skjemaproduksjon i Kostra foregår mellom 1. september og 15. oktober, med enkelte unntak for skjema med egne frister for utfylling. For hvert skjema skal det foreligge en kravspesifikasjon, og arbeidet med å utvikle skjemaene foregår etter et bestemt mønster. Først av alt skal det utarbeides en timeplan for produksjonen av skjema som fordeler både ansvar og oppgaver for de ulike leddene i arbeidet, og først når dette er avklart tas beslutninger om det faktiske innholdet i skjemaet. Det utarbeides deretter en teknisk kravspesifikasjon, og man henter frem og lagrer data som skal ligge som forhåndsutfylte felter, samt veiledningen som hører til skjemaet.

Når kravspesifikasjonen foreligger oppdateres metadatabasen med utgangspunkt i denne, og tekster på nynorsk legges til, i og med at alle skjema skal foreligge på begge målformer. Metadatabasen er en Oracle database, og skjemaene er hierarkisk oppbygget. Tilknyttet metadatabasen er en brukerapplikasjon til internt bruk for nettopp utvikling og generering av skjema i Kostra. Denne applikasjonen har et lite tilfredsstillende grensesnitt, noe som blant annet gir seg utslag i at det er ressurskrevende å vedlikeholde skjema fra år til år. Applikasjonen brukes til å vedlikeholde skjema etter krav fra fagseksjonene i SSB. Disse kravene kan for eksempel være måten spørsmålene i skjemaet er organisert på, eller det kan være ønsker om endringer i valideringer av de ulike utfyllingsfeltene i et skjema.

Hvert element i et skjema gjenkjennes med koordinater som genereres på bakgrunn av rad og kolonne. Fra skjemagenererings-applikasjonen genereres deretter en statisk HTML-fil som fagseksjonene benytter for å lese korrektur og fjerne eventuell inkonsistens som har oppstått i skjemaene. Når dette er gjort brukes en annen applikasjon for å generere XML fra HTML-filene, hvor XML-formatet som benyttes er XML4DR[2]. Sammen med en XSL-fil, som er en fast mal for alle skjema, legges skjema ut for testing på SSBs Intranett. Det er for øvrig de ulike fagseksjonene i SSB som tester sine egne skjema.

Fra skjemaene blir det deretter generert en installasjonspakke. Dette gjøres av en ekstern bedrift, Comfact AB. Disse pakkene er noe ulike avhengig av om skjema inneholder sensitiv informasjon eller ikke. Skjema med sensitiv informasjon krypteres, i motsetning til skjema med ikke-sensitiv informasjon. XML-skjemaet inneholder derfor et merke som indikerer om skjemaet skal krypteres eller ikke. Med unntak av dette er installasjonspakkene like for alle typer skjema. Pakken inneholder alle skjemaene som kommunen skal fylle ut, som hver består av en XML- og en XSL-fil. Videre inneholder pakken en flatfilskonverteringsapplikasjon, en valideringsfil, en veiledning til skjemaet i HTML, og en HTML-oversikt med linker til alle skjemaene som kommunen skal fylle ut. På dette tidspunktet i prosessen velges det også et passord til oppgavegiver, som en hash-verdi som legges inn i XML-skjemaet. Denne brukes blant annet

[2]XML4DR er en XML-spesifikasjon for elektroniske skjema utviklet gjennom et EU-prosjekt. Spesifikasjonen inneholder to hoveddeler; én datadel og én metadata-del

når data fra skjemaet skal lastes inn i databasen. Når installasjonspakken er generert sendes en feltspesifikasjon til fagseksjonene med feltnavn, datatyper og felt med koordinater. Disse feltene får deretter fagspesifikke navn, generert fra metadatabasen, og blir en semikolonseparert fil som lastes inn i metadatabasen i en spesiell mappingtabell, slik at feltene får lesbare navn. Fra den semikolonseparerte filen genereres også en fagtabell for hvert skjema og det eksporteres en tabell med metadata og valideringer til en revisjonsapplikasjon. Denne revisjonsapplikasjonen brukes av fagseksjonene for revisjon av skjema og for å legge til ekstra funksjonalitet.

3.3.2.2 Mottak av skjema

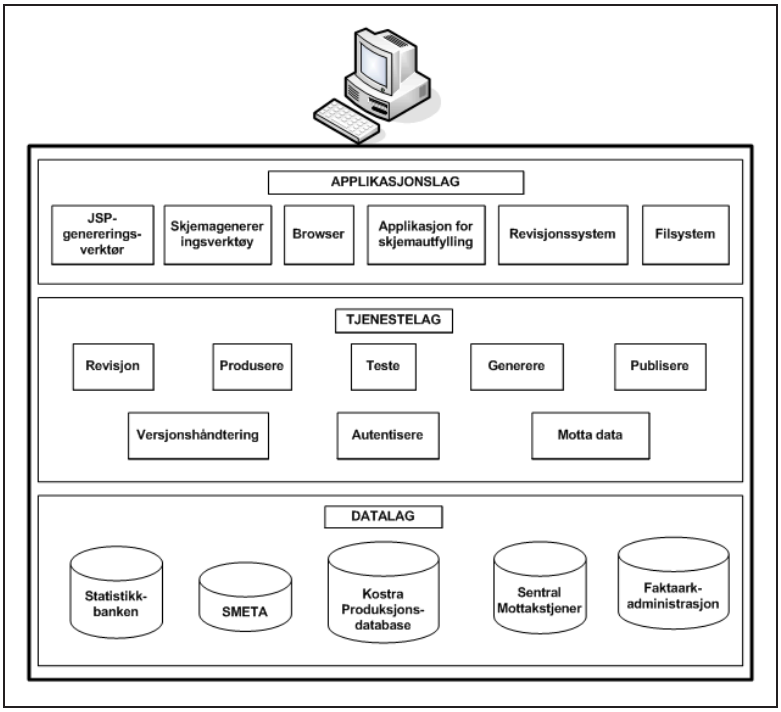
Når installasjonspakken med skjemaene er installert ute hos kommunene åpner de skjemaene, fyller inn sine data, lagrer skjemaene lokalt på sin maskin og sender det til SSB som et vedlegg i en e-post gjennom applikasjonen i installasjonspakken. Hver kommune skal i utgangspunktet ha en Kostra-koordinator som fyller ut skjemaene, men det er gjerne slik at hver etat fyller ut sin del av et skjema. Det er for øvrig gjort lite i forhold til å kartlegge kommunenes arbeid med å fylle ut skjemaene i Kostra. Noen undersøkelser er imidlertid gjort, men disse har hatt andre hovedfokus enn kommunenes organisering rundt selve skjemautfyllingen i Kostra.

Når kommunene har sendt skjemaet som e-post mottas denne av SSB. SSB bruker her procmail som ruter skjemaet, avhengig av hvilken type skjema som er mottatt. På dette tidspunktet sendes en bekreftelse til kommunen om at en e-post er mottatt. Videre i skjemamottaket sendes det imidlertid ingen tilbakemeldinger direkte til oppgavegiver om status for behandlingen av skjemaene de sender inn. Det er derfor opp til kommunene selv å sjekke at skjema de har sendt inn er mottatt og godkjente av Kostra. Dette gjøres på Kostras statussider på Internett. Skjemaet blir deretter parset, behandlingen av skjemaet loggføres og skjemaet legges i en bestemt katalog. ProSaleServer[3] poller så denne katalogen, finner filer, tolker XML, validerer data i skjemaet, dekrypterer skjema med personsensitiv informasjon, loggfører og arkiverer til slutt filene; ett arkiv for godkjente filer og ett arkiv for filer med feil. Et eget script laster deretter godkjente data inn i databasen "Sentral MottaksTjener" (SMT) og hvert femte minutt hentes skjema fra denne databasen, via metadatabasen til en faktatabell. Skjemaet går deretter igjennom et vaskeprogram som sjekker basisfunksjoner i skjemaet og gjør samtidig endringer dersom det er oppdaget feil i skjemaet tidligere, før skjemaet bygges opp steg for steg. Denne prosessen er en tung prosess, i og med at den simulerer utfylling av et skjema før det lagres i en database. Dersom skjemaet er et regnskapsskjema finvaskes dette for sjekke dubletter i posteringene og eventuelle andre endringer. For alle skjema sjekkes avgiver, og dersom avgiver har rapportert i samme periode slettes gamle data. Avgiver loggføres, og loggen kan da fortelle noe om nøkkelin-

[3]ProSale Server er utviklet av Comfact AB og håndterer innkommende og utgående data. ProSale Server kjører kontinuerlig og starter ulike aktiviteter i henhold til en gitt plan og gitte regler

formasjon i skjemaet og hva som blir primærnøkler i databasen, da dette blant annet avhenger av skjematype. Dersom avgiver har rapportert tidligere skal skjema ha samme løpenummer som tidligere, hvis ikke genereres et nytt løpenummer for det aktuelle skjemaet.

Skjemaet bygges nå opp fra bunnen, hierarkisk fra metadatabasen, og data hentes inn felt for felt. Det kontrolleres mot datatyper for metadatabasen, og for regnskap sjekkes i tillegg balansen. I dette arbeidet skrives det direkte til en feillogg og en feilloggtabell. Status for oppbygningen skrives på skjemaet og i en egen tabell; godkjent og ikke-godkjent. Deretter logges skjemaet, og det er bare godkjente skjema som legges i faktatabellen. Data blir deretter pivotert og man sjekker hvilke kommuner som har rapportert siden sist, finner avsender, henter data, og legger skjemaet i en databasetabell. Fagseksjonene reviderer så data for skjemaene og når data er godkjente av fagseksjonene overføres disse deretter til Kostras produksjonsdatabase (KPD) og til faktaarkadministrasjon hvor fagseksjonene definerer sine nøkkeltall og lager beregninger for hvert av disse innenfor gitte emner. Fagseksjonene bestemmer deretter hvor mange og hvilke typer beregninger de skal bruke. Deretter genereres et faktaark i systemet Faktaarkadmin, ett faktaark 15. mars og ett 15. juni hvert år. Det lages også en fil som sendes til Statistikkbanken.



Figur 3.4: KOSTRA arkitektur

3.3.3 Forskjeller mellom Kostra og Idun

Den kanskje største og mest åpenbare forskjellene mellom Kostra og Idun er at mens Kostra er en offline rapporteringsløsning, er Idun en

online portalløsning. Kostra har også faste rapporteringstidspunkt hvert år, mens Idun har løpende rapportering gjennom hele året. På tross av disse forskjellene, er skjemagenereringen for de to systemene relativt like. Det er allerede igangsatt et prosjekt for å samle all metadata for de to systemene i én metadatabase, hvor man tar utgangspunkt i SMETA; metadatabasen som Idun bruker i dag.

SSB har sett fordelene ved å kun ha ett rapporteringssystem å forholde seg til. Arbeidet med metadatabasen er derfor første skritt på veien mot en slik enhetlig løsning, hvor Kostra og Idun samles. Man ser i utgangspunktet for seg å bruke Idun-portalen som en basis, og flytte Kostras skjemaproduksjon og skjemamottak over til dette systemet. I utgangspunktet er det ingenting i veien for at dette skal kunne gjennomføres, da de fleste kravene til sikkerhet i Kostra også kan opprettholdes gjennom Idun. Det er imidlertid noe usikkerhet knyttet til de skjemaene i Kostra som har direkte filuttrekk fra organisasjonenes fagsystemer, og de skjema med sensitive personopplysninger som krever kryptering i sikker sone før de sendes inn til SSB. Skjemaene i Idun har per i dag ingen slike skjema, slik at det må utvikles en løsning som støtter de Kostra-skjemaene som er berørt. Skjemaene dette gjelder er for øvrig regnskap for kommune og fylkeskommune, barnevern, familievern og sosial. Dagens løsning i Kostra for disse skjemaene er at de først fylles ut i sikker sone i kommunen eller fylkeskommunen, deretter krypteres de i sikker sone, før de sendes ut av sonen og til SSB. Grunnen til oppmerksomheten rundt disse skjemaene, er at man i SSB er usikker på om det i det hele tatt finnes gode nok teknologiske løsninger som Datatilsynet vil godkjenne for sikkerhet av sensitive data i skjemaene. Rikstrygdeverket har imidlertid portalløsninger for rapportering fra leger til Trygdeetaten, og har dermed mange soner for å sikre at data i rapporteringen behandles på en tilfredsstillende måte. De fleste kommunene har i tillegg fagsystemer for enkelte data som SSB bruker for å hente data direkte, og da via en annen kanal enn Kostra. For kommunene uten slike fagsystem kan man lage spesialtilpassede løsninger, da kommunene dette gjelder for er i et lite mindretall.

3.3.3.1 Autentisering

Autentiseringsrutinene er i prinsippet nokså like for Kostra og Idun. I Idun logger oppgavegivere seg inn i portalen med organisasjonsnummer eller en lastebil- eller oljelisens, i tillegg til at de må bruke et passord generert av SSB og som følger med papirskjemaet de får tilsendt per brev. Brukernavnet og passordet er dermed det samme for hver enhet, dvs organisasjon, lastebil- eller oljelisens. I Kostra må alle avgivere benytte en entydig nøkkel under installasjonen av Kostra-pakken. Denne nøkkelen benyttes til autentisering, og av denne genereres også en hash-verdi som benyttes ved innsending av skjema til SSB.

I tillegg til å flytte Kostra over til Idun har man diskutert bruk av AltInn som rapporteringskanal til SSB. Dette krever imidlertid nye rolleidentifikatorer i AltInn, da rolleidentifikatorene per i dag bare kan knyttes mot person eller organisasjon. SSB har ikke ytret ønske om en slik

utvidelse overfor AltInn, og der foreligger dermed ingen planer om å endre dette i AltInns autentiseringsløsning.

3.3.3.2 Publisering

Både Kostra og Idun bruker Statistikkbanken etter at skjema er sendt inn til SSB. Kostra bruker i tillegg systemet Faktaarkadmin. Hit overføres data fra skjemaene, personer i de ulike fagseksjonene definerer sine nøkkeltall for de ulike skjemaene og det genereres et faktaark på bakgrunn av dette for hvert skjema. I tillegg lages det en fil som sendes til Statistikkbanken. I Idun legges data over på en Unix-server, og herfra er det oppdragsgiverne i SSB som har ansvaret for skjemaet og for å overføre data til Statistikkbanken. For å integrere de to systemene vil det derfor være naturlig å ta et valg på hvordan data i skjema skal håndteres før de skal sendes til Statistikkbanken. Man kan gjøre som man gjør i Idun i dag, hvor man legger alle data på en Unix-server, eller man kan benytte et sentralt revisjonssystem. SSB har flere revisjonssystem per i dag, noe som kommer av at de ulike fagseksjonene ved SSB selv har utviklet sine egne systemer tilpasset sine skjema.

3.3.3.3 Valg av systemløsning

Dersom man velger å bruke Iduns portalløsning på skjema i Kostra, vil det ikke lenger være et behov for statusrapporter i Kostra, da dette vil kunne vises i portalen. Mottakssystemet i Kostra vil også falle bort, da man kan benytte løsningene som ligger til grunn for Idun. Som nevnt tidligere vil dette bli en utfordring for skjema med sensitive data, men dette er ikke et større problem enn at spesialløsninger for disse tilfellene kan utvikles.

Portalløsningen i Idun er bedre enn Kostra på mange måter, både når det gjelder tilbakemelding til avgivere, og dynamiske skjema. Idun håndterer dermed også mer avanserte skjema på en bedre måte. Det er også enkelte tekniske svakheter med Kostra som fører til unødige problemer for avgivere, spesielt under installasjon av Kostra-pakken og innsending av utfylte skjema. Kostra er imidlertid bedre å på generiske kontroller, men begge systemer lider under til dels svak skjemametodikk og manglende testing av skjema.

Selv om integrasjonen mot systemene bak brukergrensesnittet for de to løsningene er ulik, er det i grove trekk kun strategiske valg som står i veien for en integrasjon av de to systemene. Teknologisk er det ingen umulighet, og ingen uoverkommelig oppgave.

3.4 Design av ny arkitektur

Tjenesteorientert utvikling er en motsetning til objektorientert utvikling, prosedyreorientert, meldingsorientert og databaseorientert utvikling. Tjenesteorientert utvikling tilbyr fordeler som gjenbruk, løse koblinger mellom teknologiske enheter og bedre ansvarsfordeling mellom de ulike enhetene i virksomheten. Utvikling av tjenester skiller seg så klart fra ob-

jektorientert utvikling fordi en tjeneste er definert ut fra meldingene den utveksler med andre tjenester, i motsetning til objektorientert utvikling hvor metodene defineres ut fra deres signatur. En tjeneste defineres med et høyere abstraksjonsnivå enn objekter fordi de skal kunne mappes til prosedyreorienterte språk som COBOL, meldingsorienterte språk som Java Message Service (JMS) eller objektorienterte systemer som J2EE.

Modellering av en tjenesteorientert arkitektur krever at man tar i bruk flere aktiviteter og artefakter enn hva som brukes i tradisjonell objektorientert analyse og design (OOAD). For det første adresserer ikke dagens OOAD-metoder det som blant mange regnes som nøkkelkonseptene i en tjenesteorientert arkitektur, nemlig tjenester, komponentene som realiserer tjenestene og meldingsflyten mellom disse (Arsanjani 2004).

3.4.1 Component and Model-based Development Methodology (COMET)

For å designe arkitektur av et integrert system av Kostra og Idun har jeg valgt å bruke utdrag av COMET, en metodologi utviklet ved SINTEF ICT (Berre, Elvesæter, Aagedal, Oldevik, Solberg & Nordmoen 2004). COMET er en objektorientert analyse- og designmetodologi, og støtter utvikling og vedlikehold av produkter og produktfamilier. At COMET er objektorientert betyr at den virkelige verden og programvaresystemer betraktes og modelleres som en mengde samhandlende objekter. COMET står dermed i kontrast til såkalte analysebaserte metodologier, som ser programvaresystemer som et hierarki av funksjoner som opererer på felles data. En viktig tanke bak COMET-metodologien er troen på at informasjonssystemer kan og bør settes sammen av komponenter.

COMET beskriver derfor metoder og teknikker som oppmuntrer til bruk av komponenter og virksomhetsobjekter som byggesteiner i forhold til å utvikle åpne, fleksible og dynamiske programvaresystemer (Berre et al. 2004). COMET er en modelldrevet metodologi, og har derfor modeller som skiller de ulike delene i utviklingen fra hverandre, og som deler opp kompleksiteten i et system. COMET tilbyr modeller av problemområdet for å øke forståelsen av hva man skal utvikle (analysemodellen), modeller av programvareløsningen for å øke forståelsen av hvordan systemet skal implementeres (designmodellen), og modeller av det kjørende systemet for å kunne øke oppdelingen av implementasjonen i henhold til logiske lag i referansearkitekturen (implementasjonsmodellen) (Berre et al. 2004).

COMET kan også sies å være en iterativ, inkrementell metodologi. Hver iterasjon produserer et brukbart system, som man utvikler og arbeider med i flere iterasjoner til resultatet er det systemet som beskrives i kravspesifikasjonen. Den inkrementelle delen av metodologien gjør at systemet deles opp i komponenter og man implementerer og integrerer systemet komponent for komponent.

COMET er use case drevet, og man kan i denne konteksten implementere et gitt system i flere steg, ett use case om gangen. Det er imidlertid viktig å huske at hvert steg av implementasjonen skal resultere i et kjørende system. COMET er også modelldrevet, og det er i hovedsak fire modell-

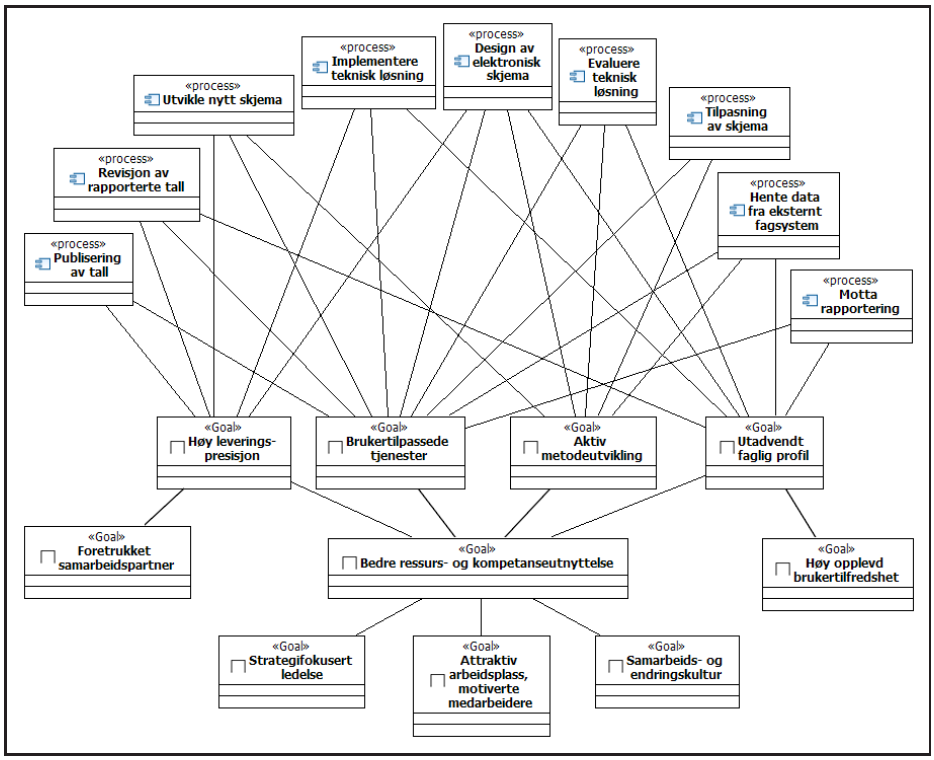
er involvert; en businessmodell, en kravmodell, en arkitekturmodell og en plattformspesifikk modell. I og med at oppgaven går ut på å designe en arkitekturmodell, vil jeg designe en såkalt plattformuavhengig modell, som uttrykkes i form av business-, krav- og arkitekturmodeller.

3.4.2 Virksomhetsmodellering

Modellene som inngår i virksomhetsmodellering er ment å uttrykke de rollene som utøves av det produktet som skal utvikles i og brukes av en gitt virksomhet, i dette tilfellet SSB. Modellene i en slik virksomhetsmodell med COMET inkluderer mål, forretningsprosesser, ulike steg i disse forretningsprosessene, roller i virksomheten og ressursene i virksomheten.

Når jeg her modellerer en systemintegrasjon av Kostra og Idun har jeg valgt å modellere målene dette systemet vil ha, hvor jeg begrenser skopet til mål og prosesser som faktisk er i bruk i systemet.

Videre har jeg valgt å modellere prosessene og rollene i SSB, i kontekst av det nye systemet, knyttet opp mot hverandre. Prosessene jeg modellerer er tatt direkte fra målmodellen, se figur 3.5 på neste side. Målene i modellene er hentet fra Statistisk sentralbyrås strategi for avdeling for IT og datafangst, og prosessene er utledet direkte fra disse i kontekst av en integrasjon av Kostra og Idun. I og med at Kostra og Idun prinsipielt sett er relativt like har ikke arbeidet med å finne gode forretningsprosesser for et integrert system vært den største utfordringen, da forskjellene for de to systemene i hovedsak ligger i implementasjonen av de ulike prosessene, og ikke i bruk av prosessene i seg selv. I arbeidet med å modellere flyten i de ulike prosessene har jeg i første rekke tatt hensyn til at de skal tjenesteorienteres og med det oppnå en god ansvarsfordeling i systemet. Et eksempel på en slik prosessmodell er illustrert i figur 3.6 på side 54 og beskriver de ulike stegene («ResourceAsArtefact») i en prosess og hva de ulike stegene produserer, hvis noe. I tillegg illustrerer modellene hvem som utfører hvert steg, om det er et verktøy involvert («Tool Step»), om steget utføres automatisk av systemet («Immediate Step») eller om det utføres av en menneskelig aktør i systemet («Human Step»). De resterende prosessmodellene for systemet er å finne i Tillegg B, fra side 117.

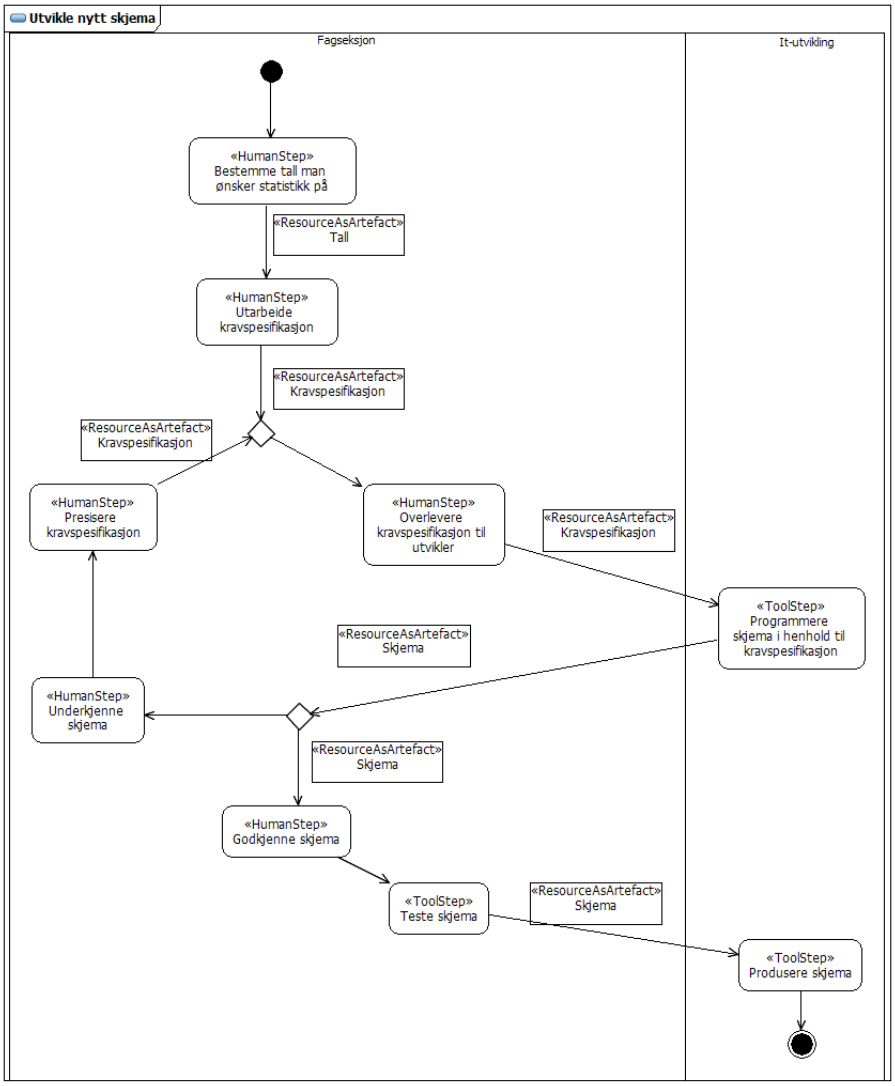


Figur 3.5: Prosesser og Mål

3.4.3 Kravmodell

Kravmodellen skal, som navnet tilsier, identifisere kravene til systemet, både de funksjonelle og de ikke-funksjonelle kravene. Ikke-funksjonelle krav er ofte vanskeligere å måle enn funksjonelle krav, og omfatter blant annet ytelse, tilgjengelighet, sikkerhet osv.

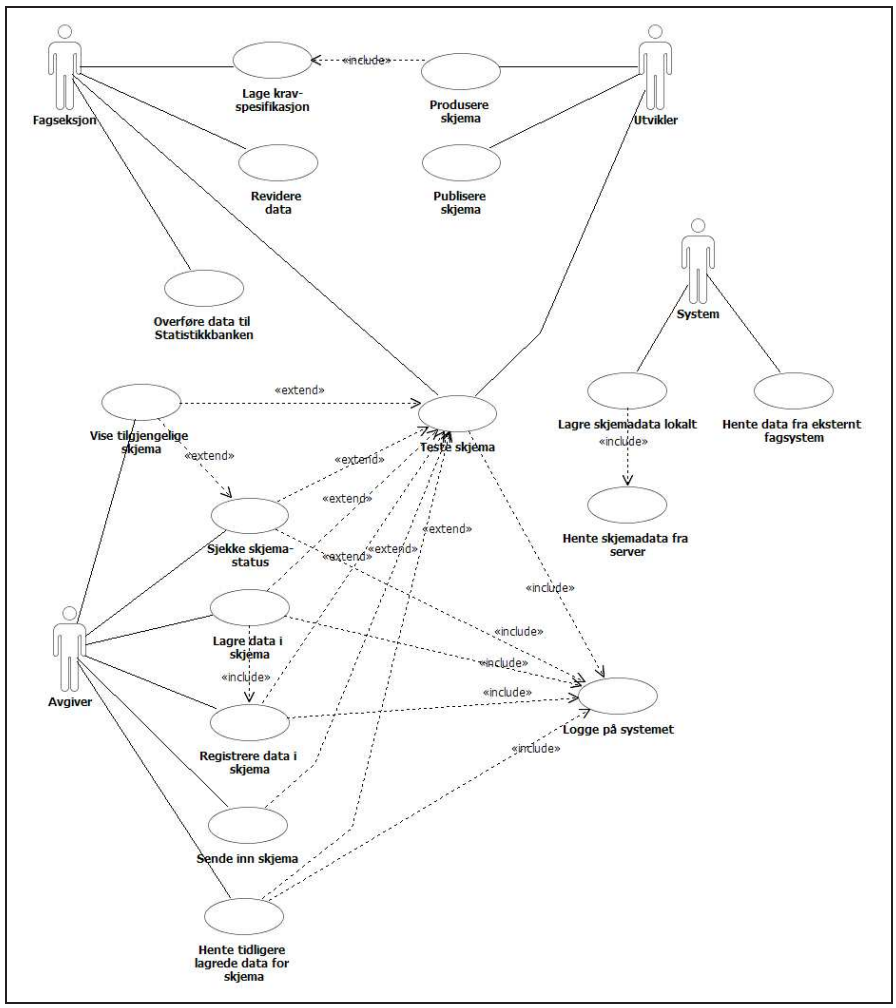
Use cases og beskrivelser av disse brukes for å fange opp de funksjonelle kravene, dvs en konkretisering av brukerne i systemet, hva systemet er ment å gjøre og hvilken funksjonalitet det skal ha. I den forbindelse har jeg modellert use cases, beskrivelser av disse og aktivitetsmodeller for hvert use case, hvor de sistnevnte modellene også er med på å beskrive meldingsflyten i systemet mellom de ulike komponentene og aktørene. Entitetene i disse modellene beskriver hva de ulike stegene i i utførelsen av use caset produserer. Tjenester og komponenter i det nye systemet introduseres for første gang i figur 3.8 på side 56, da jeg grupperer use casene med hensyn på nettopp tjenester og komponenter. Disse elementene vil også være å finne igjen i arkitekturmodellen, se figur 3.11 på side 63. En fullstendig oversikt over kravmodellene er å finne i Tillegg B fra side 125.



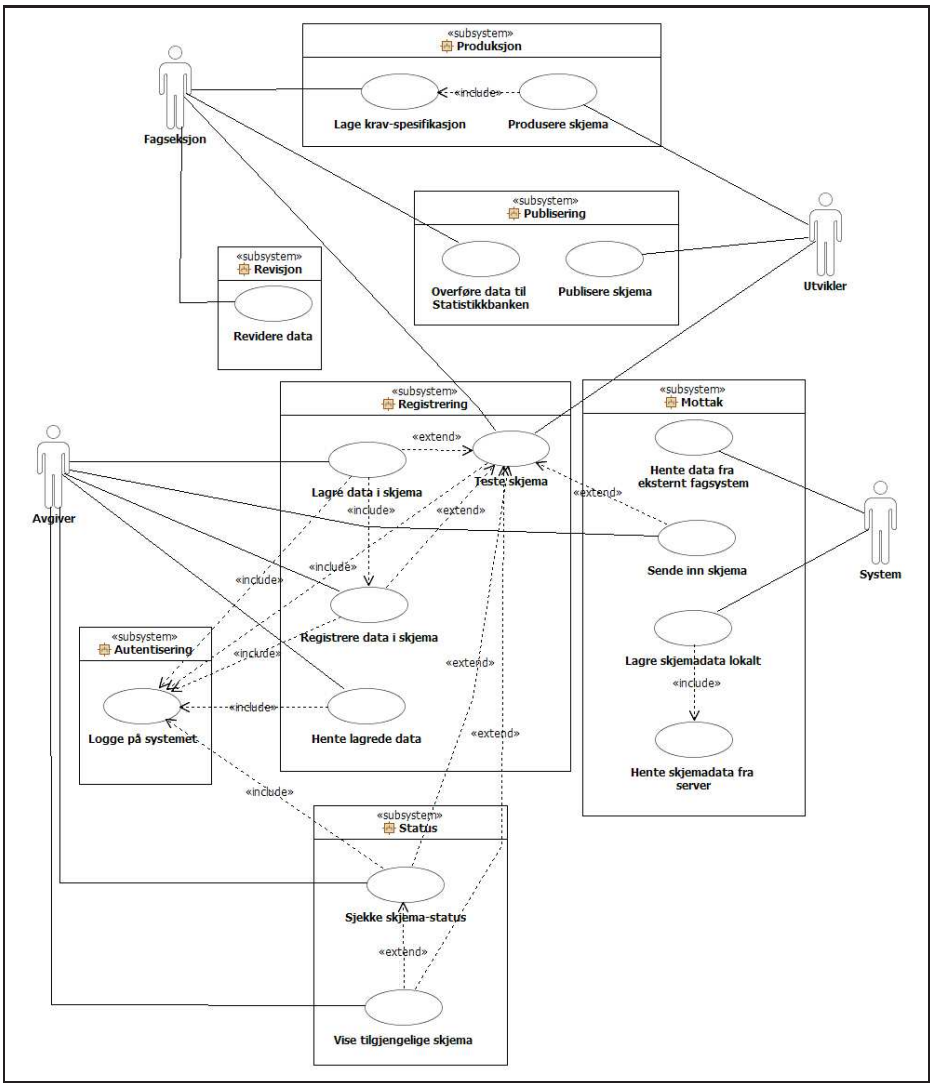
Figur 3.6: Prosessen "Utvikle nytt skjema"

3.4.3.1 Use Case Model

Som det fremgår av Use Case modellen (figur 3.7) har jeg valgt å ikke modellere bruk av faktaark som funksjonalitet i den integrerte systemløsningen av Kostra og Idun. Idun benytter i dag Statistikkbanken som publiseringskanal. Det gjør også Kostra, men som et tillegg publiserer Kostra også sine tall gjennom en egen faktaarkapplikasjon. Jeg mener det er lite hensiktsmessig å ha to publiseringskanaler for ett og samme system og for de samme dataene. Sand et al. (2006) påpeker i sin rapport at selv om Statistikkbanken mangler noe funksjonalitet i forhold til Kostras faktaarkapplikasjon er ikke denne funksjonaliteten av vesentlig karakter, og er ikke alene et godt nok argument for å beholde Kostras faktaarkapplikasjon som publiseringskanal i et nytt system. Det skal også nevnes at Statistikkbanken har funksjonalitet som Kostras faktaarkapplikasjon ikke har. Med den beslutningen blir publisering av Kostra-data mer lik publisering av data fra Idun per i dag, og Statistikkbanken blir den eneste kanalen hvor data fra en integrert systemløsning publiseres.



Figur 3.7: Use Case modell

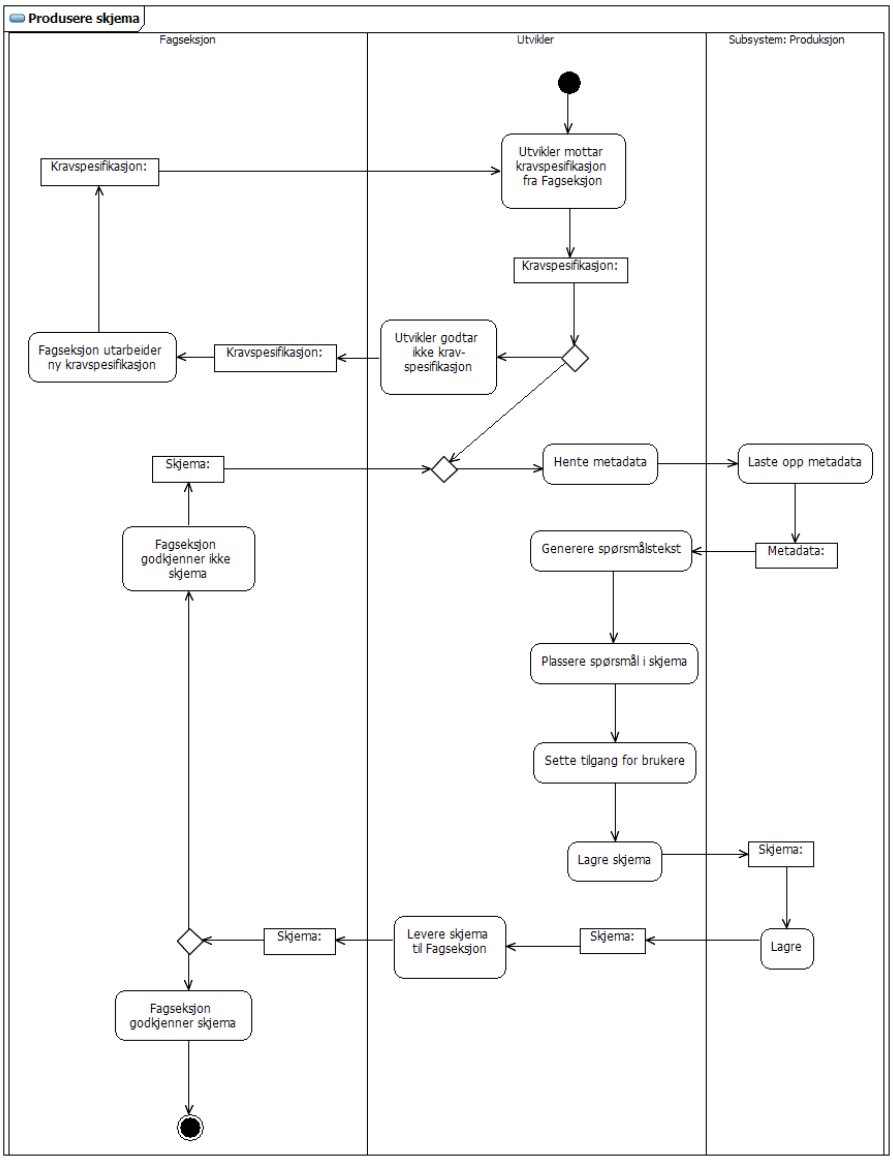


Figur 3.8: Gruppering av use cases

3.4.3.2 Use Case beskrivelser

Use Case:	**Produsere skjema**
Aktører:	Utvikler, Fagseksjon
Mål:	Produsere et nytt spørreskjema
Pre-betingelser:	Fagseksjon har dokumentert nye krav til et spørreskjema
Post-betingelser:	Spørreskjema er ferdig utviklet
Normal hendelsesflyt:	
1.	Utvikler mottar kravspesifikasjon fra fagseksjon
2.	Utvikler designer nytt skjema utfra kravspesifikasjonen
3.	Utvikler leverer skjema til Fagseksjonen
4.	Fagseksjonen godkjenner skjema
Variasjoner:	
2a.	Utvikler godtar ikke kravspesifikasjon
2a1.	Fagseksjonen utarbeider en ny kravspesifikasjon
2a2.	Use Case fortsetter fra punkt 1
4a.	Fagseksjonen godkjenner ikke skjema
4a1.	Use caset fortsetter fra punkt 2

Tabell 3.1: Beskrivelse av use caset “Produsere skjema”



Figur 3.9: Aktivitetsmodell for use caset “Produsere skjema”

3.4.3.3 Ikke-funksjonelle krav

Krav til programvaresystem klassifiseres gjerne som funksjonelle og ikke-funksjonelle krav, og skillet mellom disse kategoriene er ikke alltid like klart. Ikke-funksjonelle krav kan likevel beskrives som de kravene til systemet som ikke direkte kan settes i sammenheng med de gitte funksjonelle som systemet tilbyr, og mange slike krav refererer gjerne til systemet som helhet, og ikke til enkeltstående komponenter. Dette betyr igjen at mange ikke-funksjonelle krav ofte er mer kritiske til systemet enn de enkelte funksjonelle kravene, fordi mens konsekvensen av å ikke kunne imøtekomme et funksjonelt krav er å degradere systemet funksjonelt sett kan konsekvensen av å ikke imøtekomme et ikke-funksjonelt krav være at hele systemet blir ubrukelig.

Et vanlig problem i forhold til ikke-funksjonelle krav er imidlertid at de kan være vanskelige å verifisere, og måle. Det ideelle er derfor å uttrykke disse kravene kvantitativt slik at man enkelt kan måle om kravene er oppfylt. Målingene kan da gjøres under testingen av systemet. Under følger en liste over de viktigste ikke-funksjonelle kravene, og deretter en beskrivelse av disse:

- sikkerhet
- brukervennlighet
- oppetid
- belastning
- pålitelighet
- portabilitet

3.4.3.3.1 Sikkerhet — I og med at enkelte data som rapporteres i Kostra er sensitive personopplysninger er det er ufravikelig krav til systemet at det et sikkert i henhold til Datatilsynets bestemmelser. Det er også viktig at systemet tilbyr sikker pålogging og autentisering av alle brukere. Kravet om sikkerhet er til en viss grad også et funksjonelt krav, da man for eksempel kan sette som krav til systemet at data skal krypteres i henhold til en gitt krypteringsalgoritme, og at autentisering og autorisasjon skjer i henhold til gitte prosedyrer.

3.4.3.3.2 Brukervennlighet — Når det gjelder integrasjonen av Kostra og Idun er det viktig at systemet er brukervennlig. Med "brukervennlig" mener jeg at det skal være en lav terskel for nye brukere å ta i bruk systemet og grensesnittet skal være relativt intuitivt på den måten at det likner andre kjente kjøringsmiljø. For kravet om brukervennlighet kan man for eksempel teste dette ved at man bestemmer hvor lang tid man ideelt skal bruke på opplæring av nye brukere av systemet, for eksempel at en bruker som på forhånd er ukjent med systemet skal være selvhjulpen etter 5 timer med opplæring.

3.4.3.3.3 Oppetid — Oppetid er et annet ikke-funksjonelt krav til systemet. I og med at skjemaene i Kostra og Idun har ulike rapporteringsperioder, og at det ikke er faste perioder på året hvor det ikke rapporteres, i tillegg til at det er perioder av året hvor belastningen på systemet vil være spesielt stor, er det viktig at periodene hvor systemet er nede reduseres så mye som overhodet mulig. I tillegg bør man vurdere å eventuelt ta systemet ned for vedlikehold på tidspunkt hvor man anser ulempene for brukerne som minst mulig. Kravet om oppetid er ikke umiddelbart like enkelt å teste, men man kan for eksempel ha som mål at systemet ikke skal ha en nedetid på mer enn et gitt antall timer i løpet av ett år.

3.4.3.3.4 Belastning — Jeg var i forrige avsnitt inne på krav i forhold til at systemet skal tåle stor belastning. Man må regne med at mange, i perioder, kommer til å rapportere samtidig, dette er da også mer eller indre påkrevet for skjemaene i Kostra som har en begrenset rapporteringsperiode. For dette kravet kan man teste på hvor mange rapporteringer systemet tåler samtidig og i en gitt tidsperiode gjennom en såkalt stresstest.

3.4.3.3.5 Pålitelighet — Det er viktig at brukere av systemet oppfatter det som pålitelig. Dette er blant annet viktig i forhold til at mange skal bruke statistikken generert på bakgrunn av data innhentet ved hjelp av systemet, og de må kunne stole på at data og statistikk er korrekte. Det er også viktig at brukerne opplever at systemet er pålitelig når de rapporterer data gjennom systemet.

3.4.3.3.6 Portabilitet — Portabilitet er essensielt, spesielt når systemet er ment å skulle være en online rapporteringsløsning, da det er viktig at det fungerer like greit for de mest brukte nettlesere på markedet. Systemet skal ikke kreve at brukere anvender en gitt nettleser for å få gjennomført rapporeringen. Dette punktet blir antakelig mer og mer viktig ettersom bruk av gratis programvare og "open source"-programvare blir mer og mer vanlig. Systemet bør også kunne brukes på flere operativsystem. Kravet om portabilitet er testbart, da man kan teste om systemet fungerer tilfredsstillende for de OS og plattformer man ønsker.

3.4.3.3.7 Effektivitet — Effektivitet er et krav som ofte nevnes ved utvikling av nye informasjonssystemer. I dette tilfellet anser jeg ikke effektivitet er kritisk krav. Rapporteringen er ikke tidskritisk, men man bør likevel sette grenser for hvor lang responstid man skal kunne regne som akseptabel, og deretter teste systemet i henhold til disse grensene.

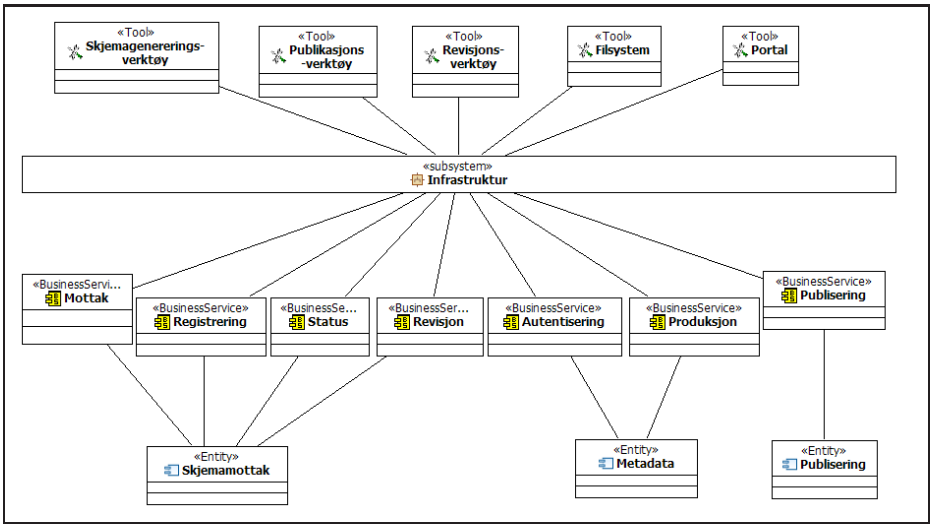
3.4.4 Arkitekturmodell

En arkitekturmodell beskriver den overordnede arkitekturen til systemet, sammen med systemets komponenter. Modellen "Komponentstrukturmodell", se figur 3.10 på side 62, beskriver systemets komponenter og deres

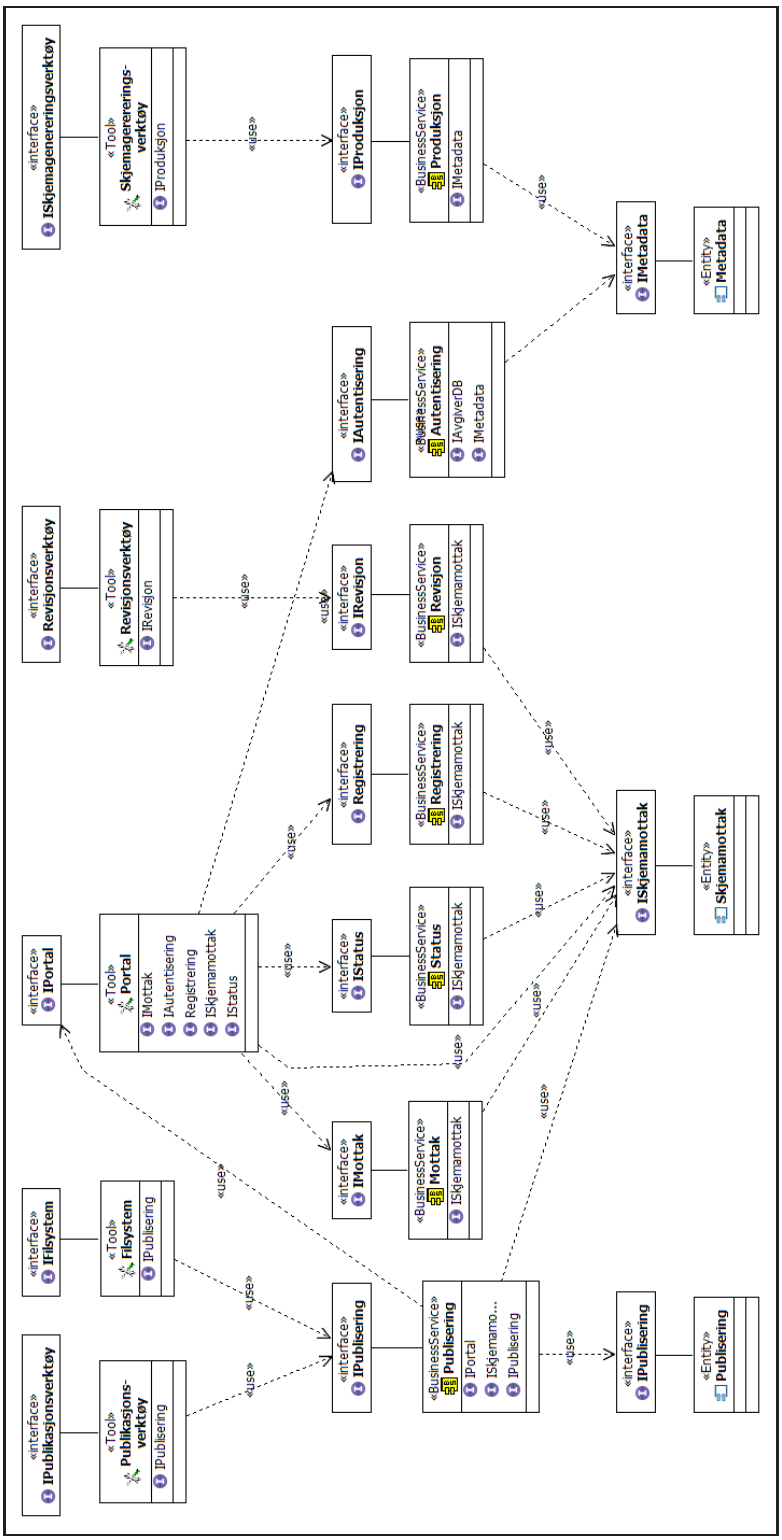
gjensidige avhengigheter Denne modellen er detaljert ytterligere i modellen i figur 3.11 på side 63 hvor også grensesnittene til komponentene er tatt med. Tjenestekomponentene jeg her har identifisert er utledet direkte fra modellen med grupperte use cases (se figur 3.8 på side 56). Verktøy jeg tenker meg brukt er først og fremst verktøy for skjemagenerering, verktøy for revisjon av mottatte data, publisering av data, et filhåndteringssystem, en portal og et verktøy for administrasjon av faktaark.

Den fullstendige arkitekturmodellen for systemintegrasjonen av Kostra og Idun er beskrevet i figur 3.12 på side 65.

3.4.4.1 Komponentstruktur



Figur 3.10: Identifikasjon av komponenter



Figur 3.11: Komponentstruktur med grensesnitt

3.4.4.2 Integrasjonsmodell

Som det fremgår av integrasjonsmodellen, se figur 3.12 på neste side, er denne delt inn i fem lag, slik Arsanjani (2004) beskriver oppbygningen av en tjenesteorientert arkitektur. Datalaget består av systemets fysiske data, representert av databasene som vil bli brukt i systemet, komponentlaget består av komponentene som realiserer funksjonaliteten i systemet, tjenestelaget inneholder tjenestene som kan brukes i systemet, applikasjonslaget inneholder de applikasjonene man kan nå tjenestene gjennom siden tjenester kan settes sammen og oppføre seg som en enkelt applikasjon (Arsanjani 2004), mens aksesslaget beskriver hvordan man får aksess til systemet.

Det er tre nøkkelelementer knyttet til en tjenesteorientert arkitektur; 1) tjenester, 2) meldingsflyt og 3) komponenter som realiserer tjenestene (Arsanjani 2004). I tillegg er det nødvendig med teknikker og prosesser for å identifisere, spesifisere og realisere tjenester med tanke på meldingsflyt og komposisjon, samt virksomhetskomponenter både for å realisere og for å sikre tjenestekvalitet. Forholdet mellom tjenester og komponenter er at virksomhetskomponentene er de entitetene i arkitekturen som realiserer tjenestene og som er ansvarlige for deres funksjonalitet og for å vedlikeholde deres tjenestekvalitet. Videre eksponeres tjenestene gjennom applikasjonene i applikasjonslaget i arkitekturen.

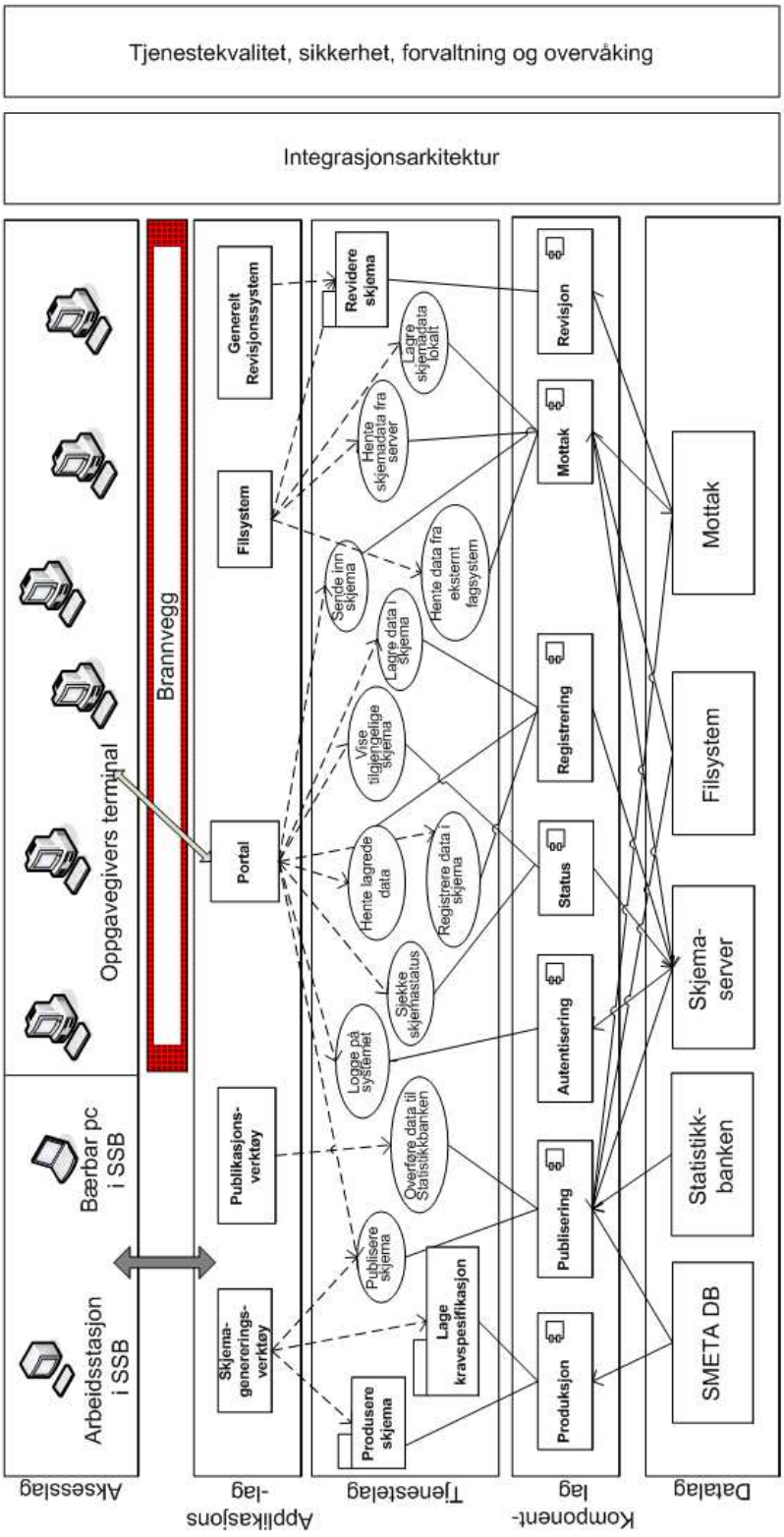
Pilene og assosiasjonene i arkitekturmodellen er ment å vise ansvarsforholdene og meldingsflyten mellom elementene i systemet. For eksempel vil et publiseringsverktøy brukes for å publisere data i systemet, i dette tilfellet gjennom tjenesten "Overføre data til Statistikkbanken". Denne tjenesten realiseres gjennom komponenten "Publiseringstjeneste" som henter data fra et filsystem og sender disse til Statistikkbanken. I en ny systemarkitektur har jeg forutsatt at det utvikles et nytt skjemagenereringsverktøy. For å unngå for mange verktøy i arbeidet med å behandle skjema, ønsker jeg å inkludere funksjonalitet for publisering av skjema i applikasjonen som brukes for å utvikle skjemaet med plassering av spørsmålstekster, innlasting av metadata osv.

Elementet "Revidere skjema", se figur 3.13 på side 66, inneholder tjenestene "Fjerne gale data", "Hente data fra filsystem" og "Sammenstille data med tidligere data". Disse er samlet i ett element, da alle disse tjenestene brukes for å revidere data.

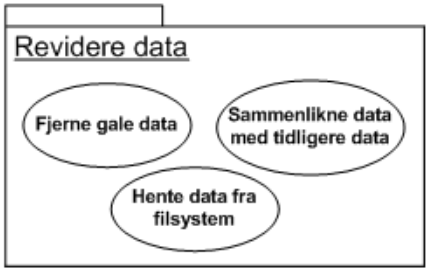
Elementet "Produsere skjema", se figur 3.14 på side 66, består av tjenestene "Laste opp metadata", "Generere spørsmålstekst", "Gi brukertilgang til oppgavegivere" og "Plassere spørsmål i skjema", som alle brukes i arbeidet med å lage et nytt skjema.

Elementet "Lage kravspesifikasjon", se figur 3.15 på side 66, består av tjenestene "Registrere krav", "Knytte krav til skjema" og "Lagre krav". Disse tjenestene er samlet i dette elementet da alle brukes for å utarbeide en kravspesifikasjon.

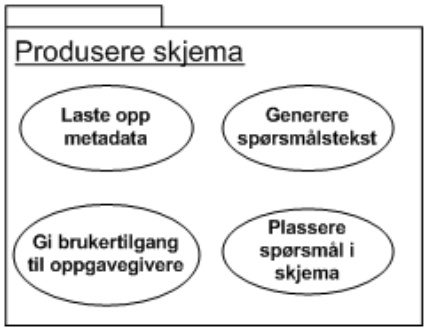
En tjenesteorientert arkitektur består av en integrasjonsarkitektur, hvis oppgave er å tilby routing, formidling og fortolkning av tjenester, komponenter og meldingsflyt i arkitekturen, blant annet ved å bruke



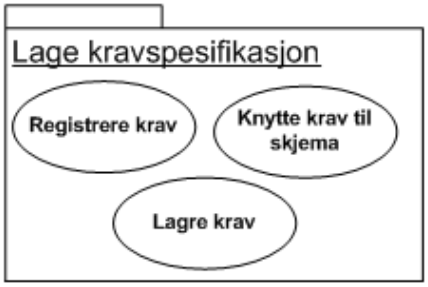
Figur 3.12: Arkitekturmodell av systemintegrasjonen av Kostra og Idun



Figur 3.13: Detaljering av elementet "Revidere skjema" i arkitekturmodellen i figur 3.12



Figur 3.14: Detaljering av elementet "Produsere skjema" i arkitekturmodellen i figur 3.12



Figur 3.15: Detaljering av elementet "Lage kravspesifikasjon" i arkitekturmodellen i figur 3.12

Enterprise Service Bus (ESB). Tjenester i bruk må videre forvaltes og overvåkes med tanke på tjenestekvalitet og sikkerhet, samt de ikke-funksjonelle kravene til arkitekturen og systemet.

Kostra behandler i dag spørreskjema med sensitive personopplysninger, og for å kunne ivareta integriteten for disse dataene er det viktig at en ny arkitektur benytter sikkerhetsmekanismer som sikrer nettopp denne integriteten. Dette kan blant annet gjøres gjennom at ulike typer data og informasjon behandles i ulike soner i organisasjonen(e), adskilt fra hverandre. Det vil uansett være viktig å ha en klar sikkerhetspolitikk for data og informasjon for å sikre at data og informasjon som sendes inn til SSB behandles i henhold til Datatilsynets retningslinjer for behandling av personopplysninger, Personopplysningsloven og for å sikre at brukerne har tillit til SSB som organisasjon.

Ikke alle mekanismer i en tjenesteorientert arkitektur er nødvendigvis standardisert, spesielt gjelder dette tjenesteorientert arkitektur med Web services og sikkerhetsmekanismene rundt denne. Fordi en tjenesteorientert arkitektur bygger på tjenester med kontrakter (se kapittel 2.2 fra side 13) som kan beskrives som veldefinerte grensesnitt, er det viktig at organisasjonen(e) arkitekturen skal implementeres i standardiserer disse så langt det er mulig for med det å sikre en enhetlig utvikling, samt enklere vedlikehold og bruk. Man bør så langt det lar seg gjøre benytte standarder under implementasjon av arkitekturen illustrert i figur 3.12 på side 65 for å enklere kunne utnytte gjenbruk av komponenter og elementer, og for å forenkle videreutvikling og unngå ad-hoc-utvikling i størst mulig grad.

Kapittel 4

Evaluering av IT-utviklingsprosjekter

Det er gjerne viktigere en man skulle tro å kartlegge status på IT-området, og det er lett å tro at man vet hva man har. I store organisasjoner hvor IT-miljøet er spredt er det imidlertid ofte slik at mange vet noe, men få, eller kanskje ingen, vet alt. Å kartlegge status er relativt enkelt. Det viser seg at det å skulle analysere uutnyttede muligheter, midler og fremtidig utvikling er langt vanskeligere (Heimly & Grøtting 2005).

Det er relativt enkelt å skulle finne kostnadene ved et tiltak. Dimensjonene som er med på å utgjøre den samlede nytteeffekten av tiltaket er vanskeligere å identifisere. Eksempler på slike dimensjoner er funksjonalitet, kvalitet, systemets nåverdi, risikobildet, omdømme og kompetanse. Effektene av tiltaket kommer gjerne over tid. Det vil si, nytteeffektene kommer gjerne over tid, mens kostnadene kommer relativt raskt. Det finnes i denne sammenheng eksempler på at prosjekter skrinlegges fordi de er vurdert over et for kort tidsintervall, og før endringene er forankret i organisasjonen og situasjonen er stabilisert.

4.1 Evalueringer av elektroniske skjema i Kostra og Idun

Det er foretatt flere evalueringer av rapporteringsløsningene Kostra og Idun de senere årene, og undersøkelsene er gjort både etter initiativ fra eksterne aktører og etter initiativ internt i SSB. I dette avsnittet følger et sammendrag av noen av disse evalueringene.

4.1.1 Evalueringer utført av SSB

To av undersøkelsene av skjema i Kostra som ble utført i 2005 var "Evalueringer av elektroniske skjema i Kostra" (Dale & Hole 2005) og "Kirkelig tjenestestatistikk i KOSTRA-drakt. Et pilotprosjekt." (Sundvoll 2005). Noen av funnene i undersøkelsene gjelder spesielt for de aktuelle skjemaene, men mange funn omhandler Kostras rapporteringsløsning

generelt, og det er naturlig å anta at funnene vil sammenfalle med andre skjema hvis disse hadde blitt testet på samme måte.

I hovedsak dreier funnene i undersøkelsene seg om nedlasting, administrasjon og distribusjon av skjema i Kostra. Administrasjonen av skjema blir gjerne veldig omstendelig, blant annet fordi Kostra ikke er en online løsning (Dale & Hole 2005, Sundvoll 2005). Kostra krever derimot nedlasting av store mengder data og informasjon, samt at det kreves et internt distribusjonssystem hos avgivere for at disse skal kunne distribuere skjema eller deler av skjema til de personene i organisasjonen som sitter inne med den aktuelle informasjonen. Nedlastingen og installasjonen av skjemaene og kontrollene i Kostra oppleves som problematisk i mange kommuner. Med en online løsning kunne deler av disse problemene vært løst. I tillegg fremmer flere kommuner et ønske om bedre fungerende verktøy. Med verktøy mener de i denne sammenhengen tekniske løsninger til hjelp under rapportering av data, i tillegg til at det er et behov for utskiftsvennlige versjoner av skjemaene, da mange bruker slike som en kladd før data fylles inn i det elektroniske skjemaet. Det er med andre ord et sterkt behov for god tilgang på tilrettelagt informasjon og hjelpeverktøy ute hos avgivere i Kostra (Dale & Hole 2005).

4.1.2 Evaluering utført av Kommunal- og regionaldepartementet

Fordi det i 2004 var flere kommuner som ikke overholdt rapporteringsfristen for Kostra-rapporteringen ønsket Kommunal- og regionaldepartementet (KRD) å finne årsakene til dette, og deretter foreslå tiltak som kan medføre en reduksjon i manglende rapportering. Undersøkelsen er gjort blant et utvalg kommuner, og det er viktig å presisere at utvalget som rapporten fra KRD er basert på (Kommunal- og regionaldepartementet 2004) ikke på noen måte er et representativt utvalg. Rapporten kan likevel illustrere kommunenes generelle holdninger og tanker om Kostra. Det er viktig å være klar over at Kostra kun utgjør en delmengde av kommunenes totale rapportering til statlige etater. Det kan likevel tyde på at statlig rapportering ikke er høyt prioritert i dagens situasjon, noe som gjenspeiles i at det for rapporteringsåret 2003 var mangler i besvarelsene fra en tredel av kommunene og en femdel av fylkeskommunene. I tillegg kom mindre enn halvparten av de nordnorske kommunene med i publiseringen fra SSB. Resultatet av dette er blant annet usikkerhet rundt makrotall fra Kostra, og det blir store skjevheter i Kostra-resultatene.

I følge rapporten er det registrert få problemer med den tekniske rapporteringen internt og med overføring av filer til SSB. Rapporten hevder likevel at flere kommuner syntes arbeidet med å legge til rette program for registrering og overføring av data var arbeidskrevende og problematisk. Hovedønsket fra kommunene slik rapporten uttrykker det er "Umiddelbar tilbakemelding/bekreftelse fra SSB på at filene er mottatt. Dersom det er feil og/eller mangler ved filene må det gis hurtig tilbakemelding og støtte/bistand til oppretting". Forsinkelsene i 2003-rapporteringen kan likevel illustreres med følgende sitat fra en kommune, hentet fra rapporten (Kommunal- og regionaldepartementet 2004): "Kostra-rapporteringen har

meget lav prioritet, anses som et onde".

4.1.3 Evaluering utført av Forvaltningsinfo AS

Forvaltningsinfo AS utarbeidet i 2003 en rapport om elektronisk rapportering i næringslivet, "IT mot skjemabelastning - elektronisk rapportering som forenklingsteknikk mot næringslivet"(Forvaltningsinfo AS 2003). Idun vil som kjent gå inn under denne beskrivelsen, og funnene i denne rapporten er derfor i høyeste grad interessante.

Forvaltningsinfo AS har spurt en del småbedrifter om deres opplevelse av offentlige skjema, hvorav tre av fire bedriftsledere helt eller delvis er enige i at offentlige skjema er et stadig irritasjonsmoment. Det er heller ikke mange i dette segmentet som mener at offentlige skjema er blitt mindre belastende de siste årene, og i underkant av 40 prosent av de spurte er skeptiske til om elektroniske skjema skal kunne lette arbeidet. En undersøkelse utført av Næringslivets Skjemaråd i 1994 viser en klar sammenheng mellom størrelsen på bedriften og dens opplevelse av tidsbruk. Paradokset her er at på tross av at mange svarer at offentlige skjema fører til "svært mye belastning", mener de samtidig at skjemaarbeidet tar mellom én og tre dager på årsbasis. Dette indikerer blant annet at tidsbruk er en utilstrekkelig parameter i forhold til problematikken rundt skjemavelde og skjemabelastning (Forvaltningsinfo AS 2003).

Næringslivet har i grove trekk tregere Internett-forbindelse enn privatpersoner (Forvaltningsinfo AS 2003), og den store andelen analoge nettilknytninger bør i høyeste grad tas i betraktning ved utvikling av løsninger for elektronisk innrapportering. Man bør derfor prioritere "lette" skjemasider fremfor "tung" grafikk i større grad enn man gjort tidligere. Gjennom intervjuer med et antall bedrifter kom det frem at irritasjonen over offentlige skjema i størst grad er rettet mot rapportering av statistikk til SSB. Forvaltningsinfo AS finner dette påfallende, fordi SSB kun står for i underkant av to prosent av den samlede tidsbruken til statlige skjema. Skatteetaten, som på sin side står for nesten 80 prosent av samlet tidsbruk, får derimot liten kritikk fra bedriftene som ble intervjuet.

4.1.4 Utfylling av skjema

Som jeg var inne på tidligere er det vesentlig å kunne redusere behovet for intern administrasjon rundt skjemautfyllingen, da mange funn i undersøkelsene relateres til klaging over høy tidsbruk i forbindelse med skjemautfyllingen. En reduksjon i administrasjonsbehovet vil samtidig kunne føre til en lavere oppgavebyrde for avgiverne i Kostra. Design er viktig for opplevd brukervennlighet, og man bør derfor ta i bruk flere virkemidler for å oppnå god skjemadesign i større grad enn man gjør i dag (Dale & Hole 2005). Dette vil også kunne være med på å lette tilgangen til nødvendig hjelp og informasjon under utfyllingen.

De automatiske kontrollene i Kostra-skjemaene er en annen kilde til problemer, og man etterlyser i mange tilfeller færre, men mer funksjonelle, kontroller. Kontrollene i Kostra-skjemaene er per i dag så mange at

advarsler og feilmeldinger ofte ignoreres under rapporteringen uten at disse blir lest, mye fordi skjema ikke fylles ut kontinuerlig og i sin helhet, men istedet fylles ut delvis hos de ulike avgivergruppene i kommunene som sitter på de aktuelle tallene. SSB bør derfor sørge for en mer bevisst bruk av kontroller i skjemaene, og man bør bruke mer tid på testing av kontrollene og en bedre kvalitetssikring i forhold til disse (Dale & Hole 2005).

Terskelen for å komme i gang med elektronisk rapportering bør for alle oppgavegivere være lavest mulig, da man verken har noen garanti for, eller kan kreve, høy kompetanse og kjennskap til teknologiske verktøy hos oppgavegiverne. Dersom terskelen er lav vil de fleste kunne rapportere sine tall direkte i et elektronisk skjema, i stedet for at ulike kommunale instanser fyller inn sine tall i et papirskjema som personer i kommunens administrasjon igjen må fylle inn i det elektroniske skjemaet ved en senere anledning, slik mange gjør i dag. Dette vil imidlertid kreve en annen oppbygning av den tekniske skjemaløsningen i dag, da flere må kunne rapportere sine data på det samme skjemaet uten at tidligere rapporterte data forsvinner. Undersøkelsen påpeker også at enkelte skjema med fordel kan deles opp i flere skjema, blant annet for å redusere antallet personer som involveres i rapportering av data for ett enkelt skjema.

Elmer-prosjektet (Enklere og mer effektiv rapportering) [1] har satt opp flere anbefalinger for hvordan Internett-skjema bør utformes for å utnytte mediets pedagogiske muligheter, og dermed fremstå mer brukervennlig. Ingen plikter å følge tankene fra Elmer-prosjektet, men Elmer-formen kan bidra til at offentlige nettskjema får et brukergrensesnitt som i størst mulig grad har lik struktur og som tar i bruk de få standard Internett-konvensjoner folk kjenner fra før.

4.2 Evaluering av IT-prosjekter

Mange mener at bedrifter og organisasjoner bruker for mange penger på IT-prosjekter uten å få tilsvarende tilbake (Remenyi et al. 2000). Organisasjoner vil gjerne se fordelene og avkastningene av IT-investeringer på samme måte som for andre typer investeringer, men det er uklart hvordan forventningene om å kunne se IT-fordeler har blitt til. Det er ofte at investeringer er gode, selv om fordelene kan være vanskelige å gripe fatt i. Organisasjoner krever gjerne at IT-gevinster uttrykkes i termer av valuta, det vil si hvor mange penger som er brukt og hvor mange penger som er spart ved å investere i et gitt IT-prosjekt. Noen investeringer kan imidlertid ikke uttrykkes i kroner og øre, selv om investeringen fører til store gevinster for organisasjonen.

Hovedgrunnen til å evaluere en systemarkitektur er for å beregne i hvilken grad systemet kan oppfylle krav til kvalitet, og for å identifisere poten-

[1] Elmer-prosjektet er et samarbeidsprosjekt mellom Nærings- og handelsdepartementet, NHO og HSH for å fremskynde bruk av elektronisk rapportering fra næringslivet. Skattedirektoratet, Rikstrygdeverket, SSB, tre departement og flere næringslivsorganisasjoner har også tatt del i prosjektet. (Forvaltningsinfo AS 2005)

sielle risiki (Zhu, Babar & Jeffery 2004). Sikkerhet er ett av kvalitetsattributtene som er vanskelige å evaluere, fordi det er vanskelig å evaluere fordelene ved de ulike, alternativene til teknologi (Butler 2002). Ledere ønsker likevel å vite om en sikkerhetsinvestering reduserer en gitt risiko til et akseptabelt nivå.

4.2.1 Investeringer

Selv om et programvareprodukt leveres i tide, på budsjett og korrekt og effektivt utfører alle krav er det ikke dermed gitt at man er fornøyd med produktet (Boehm, Brown & Lipow 1976). Dette kan komme av at krav til kvalitet ikke er oppfylt; systemet kan være vanskelig å forstå og endre, systemet kan være vanskelig å bruke og det kan være unødvendig plattformavhengig og vanskelig å integrere med andre systemer. I tillegg til å evaluere krav til kvalitet er det viktig å evaluere de rent funksjonelle kravene til et system. Disse kravene er vanligvis implementert i et subsystem eller en komponent og er dermed sporbare i arkitekturen. Kravene til kvalitet er derimot sjelden sporbare, da disse kravene gjerne er dekkende for arkitekturen som helhet (Bosch & Molin 1997).

Det er i hovedsak fire problemområder med tanke på å bestemme IT-gevinster (Remenyi et al. 2000). Disse er identifiserbare ytelsesforbedringer, systemets rekkevidde, konkrete og uhåndgripelige gevinster, og utvikling av disse gevinstene på sikt. Potensielle gevinster bør identifiseres så tidlig som mulig, men systemets miljø er gjerne så komplekst at det er vanskelig å identifisere alle fordelene på forhånd. Enkelte gevinster påvirker bare de generelle systemomgivelsene, men kan ikke identifiseres som en direkte ytelsesgevinst. Uhåndgripelige gevinster kan likevel være av stor betydning for en organisasjon. Gevinstene ved en IT-investering er heller ikke stabile over tid. Noen fordeler forsvinner, mens andre blir mer synlige og konkrete. Det er derfor vanskelig å si noe om fremtidige fordeler ved en investering, selv om dette helt klart er ønskelig.

IT-investeringer har potensielt en utledbar verdi, og IT-gevinster er ikke et teknologispørsmål i den forstand, men har med organisasjonen som helhet å gjøre. Endringer i prosesser og fremgangsmåter fører til IT-investeringer for å støtte opp om forbedringene. Dette fører igjen til forbedringer i ytelse, som kan gi økt profitt og avkastning på investeringene.

4.2.2 Evalueringsprossessen

Evaluering av informasjonsteknologi er vanskelig, men ikke desto mindre viktig å gjennomføre. Mange organisasjoner mener at 50 prosent av all investering brukes på IT, direkte eller indirekte, og det er derfor viktig å finne ut om disse investeringene gir organisasjonen ønsket avkastning. Evalueringer kan også brukes som et hjelpemiddel for organisasjoner når de er nødt til å prioritere mellom ulike IT-prosjekter. Systemdesignere må ofte velge mellom ulike systemarkitekturer fordi disse har ulik implementasjon av ikke-funksjonelle krav (Balsamo, Inverardi & Mangano 1998).

Det finnes flere ulike evalueringsteknikker, blant annet ex-ante og ex-post evalueringer (Remenyi et al. 2000). Ex-ante evalueringer er forutseende evalueringer som gjennomføres for å forutsi og evaluere innvirkningene og omfanget av fremtidige omstendigheter i organisasjonen. Ex-post evalueringer er på sin side evalueringer som fastsetter verdien av eksisterende forhold. I og med at SSB ikke har implementert en tjenesteorientert arkitektur, men ønsker å evaluere innvirkningene en slik implementasjon vil kunne gi organisasjonen, vil en ex-ante evaluering være mest aktuelt i dette tilfellet.

Ex-ante evalueringer gjennomføres vanligvis ved å benytte finansielle estimat, som enten er enkle estimater over kostnader og nytteeffekter, eller omfattende estimat med slike beregninger. Slike evalueringer krever likevel kun tilnærmede verdier for kostnader og nytteeffekter, i motsetning til ex-post evalueringer som krever nøyaktige verdier. Et eksempel på en ex-ante evalueringsteknikk er balansert målstyring.

Evalueringsprosessen er ingen enkel oppgave, blant annet på grunn av manglende konsistens i måten IT-investeringer utvikler seg på. Det sies imidlertid at det er langt vanskeligere å evaluere nytteeffekter av IT-investeringer enn av rene kostnader. Grunnene til dette er blant annet at nytteeffektene kan være mange og av svært ulik karakter. Generelt snakker man om to typer nytteeffekter; konkrete og uhåndgripelige nytteeffekter. Konkrete nytteeffekter er gevinster som påvirker organisasjonens avkastning direkte, mens uhåndgripelige gevinster har mer indirekte innvirkning på organisasjonen. Uhåndgripelige gevinster er vanskelige å måle, og det er enda vanskeligere å koble nytteverdien til en økning i lønnsomhet for organisasjonen. Det er bare på lang sikt at de totale gevinstene og effektene av et nytt IT-system eller, i dette tilfellet, en ny IT-arkitektur, vil synes, og da med bakgrunn i hvordan systemet eller arkitekturen brukes i praksis. I tillegg er gevinstene og effektene gjerne både uforutsigbare og uventede.

4.2.3 Valg av evalueringsmetode

Verdien til et aktivum i en organisasjons balanseregnskap er helt og holdent avhengig av ledelsens perspektiv eller oppfatning av organisasjonens fremtid (Remenyi et al. 2000). Kort sagt er aktivas verdier en funksjon av kontekst og oppfatning. Behovet for å måle ytelse er stort i vestlig industri, men man spør seg stadig om hva som egentlig er med på å utgjøre god ytelse for en organisasjon. For å svare på dette må man undersøke flere forhold; det vil si, man kan ikke vurdere ytelse som et isolert aktiva.

Formålet med en investering er mest kritisk i forhold til hvordan man skal velge evalueringsmetode (Remenyi et al. 2000). Dersom en IT-investering gjøres for å øke effektiviteten krever dette teknikker som arbeidsstudier eller nytte-/kostanalyser. I et annet tilfelle, dersom investeringen gjøres for å effektivisere styrings- og kontrollmekanismene, kreves det analyser av verdiskapning, aktiva for verdikjeden osv. Bakgrunnen for investeringen er altså kritisk for hvilke evalueringsmetoder og teknikker man tar i bruk. En strategisk IT-investering kan i mange tilfeller ha innvirkning på organisasjonen i forhold til å være en lav-kost-organisasjon.

Hvis strategien er basert på lav-kost bør følgende forhold tas i betraktning; først og fremst bør man vurdere om applikasjonen vil føre til direkte kostnadsreduksjoner, om arbeidskraften vil reduseres, om tiden fra utvikling til markedet vil synke, om utnyttelsen av utstyret vil bli bedre, og om produksjon i siste instans vil være godt nok.

Generelt finnes det to former for måling; fysisk opptelling og bedømmelse ved hjelp av disposisjoner, rangering og poenggiving. I tilfellene hvor det er vanskelig eller umulig å gjøre en opptelling brukes den siste teknikken. Når man skal se på uhåndgripelige gevinster ved en IT-investering kan dette gjøres i seks steg (Remenyi et al. 2000); man definerer årsaker og virkninger som kan ha kommet som en følge av det nye systemet, og man identifiserer hvordan det er mulig å fastslå endringer som kan komme som en følge av systemet. Videre bestemmer man hvordan størrelsen på endringene skal måles, før man måler størrelsen på endringene og setter en økonomisk verdi på disse ved å bruke teknikker som avkastningsgrad, netto nåverdi og internrente. Kvantitative analyser av programvaresystemer har lenge vært anerkjent i systemutviklingsmiljøer, og det er en generell oppfatning at kvantifiserbare analyser er viktige og nyttige for kravmodellering og design (Balsamo et al. 1998).

4.3 Nytte-/kostanalyser

En nytte-/kostanalyse estimerer og summerer den økonomiske verdien av fordelene og kostnadene i nåværende eller fremtidige prosjekters omgivelser for deretter å kunne bestemme om de er levedyktige (Watkins 2003). En slik analyse kan dermed brukes som et eller flere av alternativene nedenfor:

- som et planleggingsverktøy og som en hjelp til å velge mellom ulike alternativ, og for å prioritere ressurser mellom prosjekter
- som et revisjonsverktøy for å utføre evalueringer eller oppfølgende studier av eksisterende prosjekter
- for å utvikle kvantitative begrunnelser for å få politisk støtte til ulike prosjekter

Mange forfattere skiller mellom nytte-/kost- og lønnsomhetsanalyser (analyser for kostnadseffektivitet), blant annet i forhold til om nytteeffektene er kvantifiserbare eller ikke (King & Schrems 1978). Jeg, i likhet med King & Schrems (1978), kommer til å bruke "nytte-/kostanalyse" som betegnelse på begge analyseformene.

Ideen om denne typen økonomisk regnskapsførsel oppsto med en artikkel skrevet av den franske ingeniøren Jules Dupuit i 1848 (Watkins 2003). Den britiske økonomen Alfred Marshall la til noen av de formelle konseptene som i dag danner selve grunnlaget for nytte-/kostanalysen. Den praktiske utviklingen av metodikken kom likevel som et resultat av en impulshandling fra The Federal Navigation Act i 1939. Denne handlingen krevde at U.S. Corps of Engineers skulle utføre prosjekter som

skulle forbedre vannsystemet når de totale fordelene ved et prosjekt var større enn kostnadene ved prosjektet. Ingeniørkorpset laget systematiske metoder for å måle disse fordelene og kostnadene, i stor grad uten hjelp fra økonomer. Det var først mot slutten av 1950-tallet at økonomer forsøkte å opprette et konsistent sett av metoder for å måle fordeler og kostnader for å bestemme om et prosjekt er levedyktig (Watkins 2003). Noen av de tekniske aspektene ved nytte-/kostanalyser er fortsatt ikke ferdigutviklet, men fundamentet er veletablert. Nytte-/kostanalyser skiller seg imidlertid fra andre typer analyser fordi lønnsomhetskalkylen bare er en liten del av analysen, nytteeffektene vurderes i forhold til fastsatte mål for prosjektet, man vurderer forutsetningene for at gevinster kan tas ut, og man gjør en strukturert analyse av de ikke-kvantifiserbare virkningene. Resultatene av en slik analyse kan brukes både i gjennomføringen og i oppfølgingen av prosjektet analysen er gjort for, og man har en konkret plan for gevinstrealisering samt teknikker og sjekklister for å sikre at alle kostnader og nytteeffekter er tatt med.

Nytteeffekter er like viktige som kostnader når man skal foreta arkitektoniske valg i en organisasjon. Spørsmålet er hvordan organisasjonen kan investere ressursene på en måte som maksimerer utbyttet og minimerer risikoene (Kazman, Asundi & Klein 2001). For å svare på dette er det helt nødvendig med økonomiske modeller av programvaresystemer som nettopp tar hensyn til kostnader, nytteeffekter og risiki. En fullstendig nytte-/kostanalyse skal inneholde alle typer estimat for kostnader og nytteeffekter, for alle alternativ. Det er imidlertid viktig å være klar over at det er rom for subjektive vurderinger i en slik analyse (Röder & Flikke 2005).

4.3.1 IT-kostnader

Når man starter selve arbeidet med nytte-/kostanalyse er det først og fremst viktig å definere organisasjonens målsetninger. Deretter beskriver man dagens situasjon i organisasjonen, med de nåværende prosessene. Det er disse som skal danne grunnlaget for de nye, alternative prosessene. Etter at dette er gjort kan man begynne arbeidet med å definere fremtidige krav til organisasjonen. I denne sammenhengen er det viktig at man også ser kravene fra en fremtidig brukers perspektiv. Kravene er i første rekke knyttet opp mot funksjonalitet og arkitektur.

Data man samler inn vil til slutt fortelle noe om løsningens kostnader og nytteeffekter. For å få et totalt bilde av kostnadene i organisasjonen er det viktig å samle inn kostnadsdata. Disse dataene kommer gjerne fra historisk organisasjonserfaring, kostnader man har og har hatt ved bruk av dagens løsning, markedsundersøkelser, publikasjoner og analytikervurderinger (Röder & Flikke 2005). I en nytte-/kostanalyse er det viktig at man vurderer flere alternativ opp mot hverandre. Ett av disse alternativene vil til enhver tid være dagens løsning, nullalternativet. Forutsetningene for analysen må dokumenteres for å synliggjøre og forklare beslutninger gjort på grunnlag av analysen, og kostnadene til innføring av en ny løsning må selvsagt estimeres. Disse kostnadene dekker aktiviteter og ressurser knyttet til den nye løsningen, samt ulike kostnadskategorier.

Når man gjør arkitekturanalyser ønsker man primært å undersøke hvor godt arkitekturen er designet med tanke på kvalitetsattributter som ytelse, endringsdyktighet, tilgjengelighet og brukervennlighet. Kostander knyttet til kvalitet kan deles inn i to grupper; kostnader til tilpasninger av systemet og kostander til feilretting og avviksbehandling (Slaughter, Harter & Krishnan 1998).

Det er et kjent faktum at IT-kostnader er vanskelige å beregne, samtidig som mange mener at IT-kostnadene bør håndteres og styres mer effektivt enn hva som er tilfelle i mange organisasjoner i dag (Remenyi et al. 2000). På den måten har også forskere blitt mer interessert i hvordan IT-kostnader oppstår, hvordan de planlegges, styres, overvåkes og kontrolleres. Som en start i arbeidet med å få oversikt over en organisasjons IT-kostnader kan man svare på tre forholdsvis enkle spørsmål (Remenyi et al. 2000); hva organisasjonen skal benytte IT til, hvorfor organisasjonen skal gjøre dette og hvordan organisasjonen utnytter de ulike bestanddelene av sine IT-ressurser?

Kostnadsestimering av programvare handler i stor grad om å forutse de ressursene som kreves for utviklingsprosessen. Komponenter i denne sammenhengen er hardware og software kostnader, kostnader som følge av reiser og opplæring, samt kostnader som følge av innsatsen til de som er involvert. Når man snakker om å prise programvare er det flere faktorer som spiller inn. Faktorene kan blant annet være organisasjonens markedsandel, usikkerheten i kostnadsestimatene, vilkår i utviklingskontrakten, ustabile krav og finansielle sikkerheter. Produktivitet er et viktig attributt ved kostnadsestimat, og i denne sammenhengen er det viktig med erfaring både med applikasjonsdomenet, prosesskvalitet, den gitte prosjektstørrelsen, teknologistøtte og arbeidsmiljøet. Man må likevel være klar over at alle parameterne basert på volum eller tid til dels er feilaktige idet de ikke tar hensyn til kvalitet. Produktivitet blir på mange områder økt på bekostning av kvalitet.

4.3.2 Gevinster

Ved vurdering av en gevinstrealisering ser man aller først på om organisasjonens tradisjonelle ressursbruk kan reduseres som følge av innføringen av en ny arkitektur, og om noe av denne ressursbruken kan fjernes fullstendig. Deretter ser man på om medarbeidere etter hvert kan overføres til andre oppgaver som følge av mindre ressursbehov, og om endringene i arbeidsprosessene kan føre til mer effektivt arbeid. En nytte-/kostanalyse er en analyse av muligheten for innføring av nye løsninger ved å demonstrere fordelene ved kostnadsbesparelser med tanke på å oppnå styring og støtte til nettopp å implementere en eller flere av de nye løsningene som inngår i analysen.

Når man vurderer gevinstpotensialet er det viktig at man fokuserer på organisasjonen som helhet, slik at man oppretter en bevisst strategi for hvordan effekter skal tas ut. Dette oppnår man ved å sammenlikne konkret kunnskap om ressursbruk og dagens arbeidsmåter med nye prosjekter og teknologiske alternativ.

Under normale omstendigheter vil en organisasjon gjøre en investering utelukkende dersom man anser at kapitalavkastningen er tilfredsstillende for organisasjonens langsiktige mål og behov; det vil si dersom gevinsten man oppnår er verdt utgiftene som investeringen krever (Remenyi et al. 2000). Det er viktig at man ikke ser på IT som et ubetinget gode, eller en ulempe, i seg selv, og på den måten godtar at gevinsten ved en slik investering ikke ble så stor som den kanskje burde for å veie opp for de totale kostnadene. Et annet poeng i denne sammenhengen er at risiki knyttet til beslutninger om IT er mye større enn ved mange andre typer investeringer, slik at forventet gevinst bør være stor før en investering kan rettferdiggjøres (Remenyi et al. 2000).

Det er ikke uvanlig at prosjekter hvor gevinstene ennå ikke er synlige har negativ netto nåverdi. I slike tilfeller må man kunne anta at de ikke påkostede uhåndgripelige gevinstene er antatt å skulle gjengjelde utgiftene. IT-bestemmelser kan generelt sett beskrives av følgende karakteristikker; det er gjerne flere valgmuligheter knyttet til en investering, beslutningstakere forstår ofte ikke detaljene i prosjektet, kostnadene er ikke intuitive og tilliten til estimatene er liten, det er vanskelig å estimere fordelene og det er knyttet liten tillit til estimat basert på tidligere erfaringer. I tillegg er kostnadene i forhold til å ikke forplikte seg til prosjektet vanskelige å forstå eller akseptere, og uhåndgripelige gevinster er vanskelige å kvantifisere selv om de er viktige for begrunnelsen om å investere eller ikke. Det er viktig å konkretisere nytteeffektene i en slik analyse, fordi vage definisjoner av typen "forbedret ytelse" uten noen nærmere forklaring i praksis ikke har noen hensikt.

4.3.3 Cost Benefit Analysis Model, CBAM

Architecture Tradeoff Analysis Method (ATAM) er et hjelpemiddel for å forstå konsekvensene av arkitekturmessige beslutninger med tanke på kvalitetsattributter og forretningsmål (Lee & Choi 2005). Et kvalitetsattributt kan best beskrives som et ikke-funksjonelt krav (Zhu et al. 2004). Output fra denne metoden er en beskrivelse av nivåene i organisasjonen som er avgjørende for systemet, et sett arkitektoniske utsnitt som dokumenterer eksisterende og/eller fremtidig arkitektur, en oversikt over aktørenes mål i arkitekturen, risiki som er identifisert, sensitive punkter som påvirker flere kvalitetsattributter og et sett med punkter som påvirker andre kvalitetsattributter, både positivt og negativt. ATAM er imidlertid til liten hjelp for økonomiske analyser. Cost Benefit Analysis Modell (CBAM) tar over der ATAM slutter, og fungerer som et hjelpemiddel for å finne og dokumentere kostnader, nytteeffekter og usikkerhet, for på den måten å utarbeide et beslutningsgrunnlag (Kazman et al. 2001, Lee & Choi 2005).

Det er en stor grad av usikkerhet i forhold til å designe store komplekse systemer med flere aktører involvert. Usikkerhet knyttes gjerne opp mot hvordan arkitekturen forholder seg til kvalitetsattributtene, hvordan arkitekturen forholder seg til kostnadene og hvordan den forholder seg til nytteeffektene.

CBAM består av 7 steg:

1. **Valg av scenario og arkitektoniske strategier** — for hvert scenario av en viss betydning bør man beskrive en strategi for endringen

2. **Angi nytteeffekter ved kvalitetsattributtene** — nytteeffekter bør samsvare med i hvilken grad strategien støtter målene for hvert kvalitetsattributt, som i sin tur refererer tilbake til målene for organisasjonen. Kvalitetsattributtene får dermed poeng slik at man lettere skal kunne beregne deres verdi

3. **Kvantifiserbare nytteeffekter** — poengene fra forrige punkt brukes for å evaluere hver av strategiene. Alle aktørene er med på å gi kvalitetsattributtene poeng, og det er viktig å huske at aktørene sannsynligvis vil evaluere suksess og nytteeffekter ulikt

4. **Kvantifiserbare kostnader** — man kalkulerer de antatte kostnadene ved å implementere hver av strategiene som resulterer i en gitt nytteeffekt

5. **Kalkulere nytteeffekt** — man bruker gjerne denne beregningen for å rangere strategiene:

$$\text{Ønskelighet} = \frac{\text{Nytteeffekt}}{\text{Kostnad}} \tag{4.1}$$

   hvor Ønskelighet > 1 for at strategien skal være lønnsom. Denne beregningen kalles også prosjektets nytte-/kostnadsverdi

6. **Ta avgjørelser** — resultatene for beregningene plottes inn i en graf, hvor nytteeffektene utgjør y-aksen og kostnader utgjør x-aksen

7. **Analysere data** — det er viktig å måle usikkerhet ikke bare i forhold til økonomiske elementer, men også i forhold til ytelse

En nytte-/kostanalyse kan på mange måter sies å være en lønnsomhetsvurdering med det mål å gi en god oversikt over fordeler og kostnader ved å innføre nye forretningsprosesser. En slik analyse vil også danne et beslutningsgrunnlag for hvilke tiltak eller prosjekter som skal gjennomføres i organisasjonen. Analysen i seg selv skal klart angi de underliggende forutsetningene som sammen danner grunnlaget for resultatet av analysen, og løsningen som velges skal baseres på tallene fra nytte-/kostanalyse; fordelene ved systemet skal overstige kostnadene i systemets totale livsløp.

Bakgrunnen for å gjøre en slik analyse er at det er viktig å få frem og synliggjøre fordelene ved innføringen av et nytt prosjekt, ikke bare kostnadene, selv om disse gjerne er mest fremtredende, spesielt i et prosjekts oppstartfase. En nytte-/kostanalyse er derfor også en forberedelse på gevinstrealisering i organisasjonen, og mange mener en slik analyse er en av de mest strategiske måtene å støtte opp om organisasjonens forretningsprosesser på.

4.3.3.1 Viktigheten av en nytte-/kostanalyse

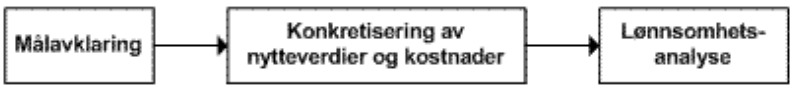
Offentlige ressurser er knappe, og mange gode formål konkurrerer om tilgjengelige midler. At konsekvenser av alternative tiltak er undersøkt og godt dokumentert er derfor en helt grunnleggende forutsetning for en fornuftig prioritering (Hervik, Hagen, Nyborg, Scheel & Sletner 1997). Det er samtidig viktig å huske at det å holde seg til dagens situasjon, nullalternativet, alltid er å regne som ett av alternativene ved en evaluering. Hovedmålet med en nytte-kostanalyse er derfor å kartlegge og synliggjøre konsekvensene av alternative tiltak. Det er vanskelig å kvantifisere tjenester i offentlig sektor. Dette kommer blant annet av at det ikke finnes et reelt press i forhold til å gjøre en gitt tjeneste mer effektiv og rasjonell, fordi det ikke finnes noen reell konkurranse til den aktuelle offentlige tjenesten.

Nytte-kostanalyser er en måte å systematisere informasjonen på, og de har samtidig en form som gjør det mulig å etterprøve de enkelte forutsetningene som et vedtak er bygget på. I en nytte-/kostanalyse verdsettes alle effekter i kroner og øre, for på den måten å enkelt kunne veie betydningen av de ulike effektene opp mot hverandre. At noe er "samfunnsøkonomisk lønnsomt" betyr at befolkningen til sammen er villige til å betale minst så mye som tiltaket faktisk koster. Dette betyr ikke nødvendigvis det samme som at tiltaket er "ønskelig" sett fra samfunnets synsvinkel, og i praksis er det derfor slik at andre hensyn enn betalingsvillighet vil veie tungt i den samlede vurderingen av en sak (Hervik et al. 1997).

Nytte-/kostanalyser gjennomføres gjerne før man tar en eventuell beslutning om å sette i verk et tiltak. Nytte-/kostanalyser kan også gi verdifull styringsinformasjon for å sikre god gjennomføring av det aktuelle tiltaket, og man kan derfor også benytte analysen etter at prosjektet er igangsatt. Kostnadsbegrepet i samfunnsøkonomiske lønnsomhetsanalyser bygger gjerne på det man kaller alternativ kostnadstankegang, som går ut på at ressursene er begrensede og at de derfor har én om ikke flere alternative anvendelser. Dette medfører at ressursene brukt i ett prosjekt alternativt kunne vært benyttet på en annen måte, for eksempel i et andre prosjekter.

4.3.4 Gjennomføring av en nytte-/kostanalyse

Statskonsult, (Lien & Bjørn 1990), beskriver dette prosessløpet for en nytte-kostanalyse, se figur 4.1.



Figur 4.1: Prosessløp for en nytte-/kostanalyse, hentet fra Lien & Bjørn (1990)

I målavklaringen konkretiserer man de resultatene prosjektet skal føre

til. Dette kan for eksempel være kostnadsreduksjoner, økt produksjon, bedre styringsmiljø, eller nye og bedre tjenester. Konkretisering av nytteeffekter og kostnader handler om at man skal liste opp alle forventede virkninger, kvantifisere disse så langt dette er mulig, og klargjøre de forutsetninger man legger til grunn for de ulike virkningene. I selve lønnsomhetsanalysen regner man ut nåverdien for prosjektet, i tillegg til at man gjerne utfører en følsomhetsanalyse, samt en analyse av budsjettmessige virkninger og de ikke-kvantifiserbare virkningene.

Det er viktig å fokusere på dagens situasjon, slik at tidligere kostnadsbesparelser og effektivitetsgevinster ikke tas med i analysen (Röder & Flikke 2005). Det samme prinsippet gjelder også for tidligere kostnader man har hatt i organisasjonen. Når man skal gjennomføre en nytte-/kostanalyse er det viktig at man først beskriver organisasjonens målsetninger samt at man beskriver de nåværende prosessene i organisasjonen.

Etter at dagens situasjon er kartlagt bør man samle og estimere kostnadsdata i forhold til å utvikle en ny løsning. Disse kostnadsdataene bør blant annet baseres på fremtidige investeringer og anskaffelser som følge av den nye løsningen, eventuelle økte driftskostnader som følge av dette, eventuelle nye lisenser som må skaffes til veie, samt kostnadene en eventuell organisasjonsendring vil føre med seg. Som nevnt tidligere er det viktig å synliggjøre fordelene ved implementering av nye løsninger. Nytteeffektene ved en slik innføring kan være av både kvantitativ og kvalitativ karakter. De kvantitative effektene er målbare effekter som økonomiske besparelser, økt arbeidskapasitet i organisasjonen og lavere bemanningsbehov. Kvalitative effekter er til gjengjeld ikke like enkle å måle. Eksempler på kvalitative effekter er forbedret kvalitet på tjenester organisasjonen leverer, bedre service til kunder av organisasjonen og bedre tilgjengelighet for brukere av tjenestene. Når man skal finne nytteverdier i et prosjekt er det samtidig viktig å fokusere på de nytteeffektene som faktisk lar seg realisere, og det er viktig at de nytteeffektene som ikke lar seg kvantifisere får en presis verbal beskrivelse, slik at de enkelt lar seg etterprøve, og slik at man enklere kan ta ut gevinsten av disse.

4.3.5 Estimat

Man skiller i grove trekk mellom to typer estimat; top-down og bottom-up estimat (Sommerville 2001). Ved top-down estimat tar man hensyn til kostnader som integrasjon, konfigurasjonsstyring og dokumentasjon. Denne teknikken kan brukes uten at man kjenner til systemets arkitektur og komponentene i denne. Et bottom-up estimat er en passende metode dersom systemet er designet i detalj. Teknikken er nyttig når systemarkitekturen og dens komponenter er kjent (Sommerville 2001). I og med at det alltid vil være usikkerhet knyttet til estimat er det viktig at man bruker flere ulike metoder på samme prosjekt, og baserer estimatet på bakgrunn av et samlet resultat.

Selv om kostnader ofte antas å være enklere å estimere enn gevinstene er ikke dette alltid tilfelle (Remenyi et al. 2000). Den største faren er at man har lett for å underestimere, gjerne fordi ledelsen i organisasjonen ikke fullt

ut forstår de antakelser som må gjøres og avhengigheter mellom disse. Det er imidlertid viktig å være klar over at det kan være både politiske og organisatoriske årsaker til at man velger å underestimere kostnader.

4.3.5.1 Systemkostnader

Estimat over systemkostnader begrunnes ofte av systemarkitekturen og inkluderer ytelse av maskin- og programvare, transaksjonsvolum, deling av ressurser på terminaler, nettverk og lignende, tilleggsfunksjoner til en gitt bruker (jf. sikkerhetsmekanismer), designfaktorer som kan beskytte ytelsen på sikt, løpende driftsutgifter, balansen mellom utvikling- og vedlikeholdskostnader, de svakeste leddene i nettverket, nettverksarkitekturen, servere og nettverkssikkerhet som brannvegger og liknende. I tillegg må man overveie om kostnadene skal inkludere indirekte kostnader, og hvordan disse eventuelt skal måles og regnes med.

4.3.5.2 Direkte kostnader

Ettersom hardwarekostnader synker er tendensen at menneskelige og organisatoriske kostnader øker (Remenyi et al. 2000). Disse kostnadene kan være opp til fire ganger høyere enn direkte kostnader knyttet til prosjektet. Direkte kostnader er kostnader som kan tilskrives implementasjon av og operasjon på ny teknologi. Disse kostnadene inkluderer også uventede behov for tilleggsutstyr. I tillegg er installasjon og konfigurasjon direkte kostnader og inkluderer for eksempel konsulenttjenester, installasjoner og oppsett av maskin- og programvarenettverk.

4.3.5.3 Indirekte kostnader

Det har vist seg at de indirekte kostnadene ofte er av langt større betydning enn de direkte (Remenyi et al. 2000). I og med at slike kostnader er så vanskelige å identifisere blir de ofte ikke tatt i betraktning når avgjørelser skal tas. Indirekte kostnader deles gjerne inn i menneskelige og organisatoriske kostnader, hvorav den største menneskelige kostnaden er den som utgjør tiden brukt på ledelse og styring, for å integrere nye systemer i eksisterende arbeidsrutiner. Andre viktige kostnadsfaktorer for å indikere menneskelige kostnader er brukerstøtte, systemstøtte og feilsøking i systemet. Andre indirekte kostnader kan være økte lønninger blant de ansatte som følge av kompetanseheving, i tillegg til kostnader ved eierskap, personalhåndtering, motivasjon, innsats i ledelsesspørsmål og ledelsesressurser.

4.3.5.4 Måling av verdier

Ett av problemene knyttet til en nytte-/kostanalyse er at selv om målingene for mange komponenter av fordeler og kostnader er relativt åpenbare, er det andre målinger hvor metodene for å utføre disse ikke er like intuitive. Først og fremst er det viktig at alle aspekter av et prosjekt uttrykkes gjennom en felles enhet, og i de aller fleste tilfeller er denne enheten penger.

På sikt er det derfor viktig å ta hensyn til pengenes verdi; dagens krone har neppe den samme verdien nå som noen år frem i tid. Verdien av fordeler og kostnader skal reflektere prioriteringer gjennom de valg man har tatt. Det er også viktig å merke seg at minste mulige verdi av fordelene ved et prosjekt er lik markedsprisen for produktet, dvs. kostnadene en eventuell kjøper er villig til å betale. På samme måte vil derfor også minste mulig verdi av fordelene minke ettersom markedsprisen må synke for å få forbrukere til å kjøpe større mengder av den aktuelle varen. Fordelene ved et prosjekt blir dermed differansen mellom hva situasjonen ville vært med og uten prosjektet. Dersom forespeilet verdi av fordelene overskrider de forespeilede kostnadene er prosjektet levedyktig.

4.3.5.5 Nåverdiberegning

Når kostnadene er estimert bør man vurdere mulige nytteeffekter som resultat av innføring av en ny løsning. Her bør man vurdere hvilke muligheter og kvaliteter som er knyttet opp mot hvert enkelt alternativ i analysen. Mange nytteverdier ved innføring av en ny løsning er som nevnt ikke umiddelbart målbare parametere, men snarere kvalitative verdier. For å kunne få disse med i beregningen er det viktig at man gir disse tallverdier slik at også disse kan brukes til sammenlikningsformål. Ved en sammenlikning av alternativ er det viktig at alle alternativene er gitt de samme forutsetningene. Hvis ikke blir sammenlikningen unøyaktig, og kanskje til og med feil. Når alle data er brakt på det rene bør man beregne kostnader og nytteeffekter for hvert år i systemets levetid for hvert av alternativene.

De ulike kostnads- og nytteelementene oppstår som nevnt sjelden på samme tidspunkt. Derfor trenger man en metode for å kunne summere nytteeffekter og kostnader, målt i kroner og øre, som påløper hvert år. Den vanligste metoden for å kunne gjøre den slik sammenlikning er å regne om de årlige nytte- og kostnadselementene til netto nåverdi.

$$NNV = I_0 + \sum_{i=1}^{n} \frac{U_i}{(1+k)^i} \tag{4.2}$$

Hvor NNV er netto nåverdi, I_0 er investeringsutgiften i år 0, U_i er nytteoverskuddet i år i, k er diskonteringsrenten som man må forutsette at er konstant i analyseperioden og n er det antall år man beregner kostnader og nytteeffekter for (Hervik et al. 1997). Ved utregning av nåverdien til et prosjekt må man derfor bestemme prosjektets levetid, for hvilket år de ulike nytte- og kostnadsverdiene skal føres opp, når på året effektene oppstår, hvilket prisnivå som gjelder for den aktuelle perioden, hvilket år man diskonterer i forhold til, hvilke kostnader og nytteeffekter som skal føres opp og hvilken kalkulasjonsrente som skal benyttes i beregningen (Lien & Bjørn 1990). Statskonsult, (Lien & Bjørn 1990), hevder samtidig på generelt grunnlag at IT-prosjekter bør vise lønnsomhet innen en 5-års periode.

Man kan også beregne lønnsomhetsnivået ved hjelp av nyttekostnadsbrøken hvor

$$N/K = \frac{\text{Nåverdien av brutto nytte}}{\text{Nåverdien av brutto kostnader}} \tag{4.3}$$

(Hervik et al. 1997).

Om man nå skulle gjennomført en følsomhetsanalyse ville man gjort endringer i forutsetningene for prosjektet og se hvilke utslag dette ville gitt for de ulike alternativene. Disse endringene kunne blant annet vært optimistiske versus pessimistiske anslag dvs "worst-case" og "best-case" scenarier, tidsforskyvinger for gevinster og kostnader, endringer i kravene til avkastning av prosjektet og endringer i ulike faktorer som fravær av innsparinger.

4.3.6 Nytte-/kostevaluering av investeringer i en tjenesteorientert arkitektur med Web services

Å utvikle en Web service er et flerårsprosjekt (Larsen & Bloniarz 2000). Organisasjoner bør planlegge for nok ressurser for å opprettholde og forbedre en tjeneste så snart den er utviklet. Kostnadene av de menneskelige ressursene som skal utvikle og kjøre en Web service er vanligvis mye større enn kostnadene til tekniske komponenter, fordi Web services teknologi kan involvere store deler av organisasjonen bør utviklingsteamet bestå av alle parter som kan bli berørt av tjenestene som utvikles. I situasjoner hvor den beste tekniske infrastrukturen ennå ikke er på plass må man sette av midler til en foreløpig infrastruktur i tillegg.

Det kan være vanskelig å identifisere effektene av en tjeneste i forhold til andre faktorer, men man finner i hovedsak fem kostnadskategorier (Larsen & Bloniarz 2000):

1. utvikling og vedlikehold
2. organisatoriske ferdigheter
3. infrastruktur
4. tilgang for personale/ansatte og andre brukere
5. sluttbrukerstøtte

Utvikling og vedlikehold krever som et minimum at eksisterende informasjon konverteres til en syntaks som kan leveres av web serveren. Organisatoriske ferdigheter har med å gjøre i hvilken grad organisasjonen er klar for å gjøre nytte av Web services. Dette kan variere veldig fra organisasjon til organisasjon. Infrastruktur betyr at tjenesten må være tilgjengelig via Internett og at en slik tilkobling må være mulig så snart tjenesten er ferdig utviklet. Tilgang for ansatte og andre brukere betyr at mens noen er med på å utvikle en tjeneste, skal andre bruke den og andre igjen skal tilby teknisk support for tjenesten. Alle disse gruppene trenger tilgang til ulike typer teknologi. Sluttbrukerstøtte vil si at organisasjonens

brukere og eksterne brukere antakelig vil ha behov for opplæring og helpdesk-støtte for å kunne bruke tjenesten effektivt.

Andre utfordringer med utvikling av en tjenesteorienert arkitektur vil typisk være opplæring av brukere og ansatte til riktig bruk av den nye arkitekturen, og det vil kunne bli en utfordring å finne gode design patterns og fullverdige modelleringsverktøy. Til slutt kreves det ofte, under utvikling av tjenesteorienterte arkitekturer, at utviklerne har bedre kunnskaper om brukerproblemer og kunnskapsdomenet i virksomheten man utvikler arkitekturen for, enn hva man er vant med fra andre utviklingsprosjekter (Král & Žemliča 2004).

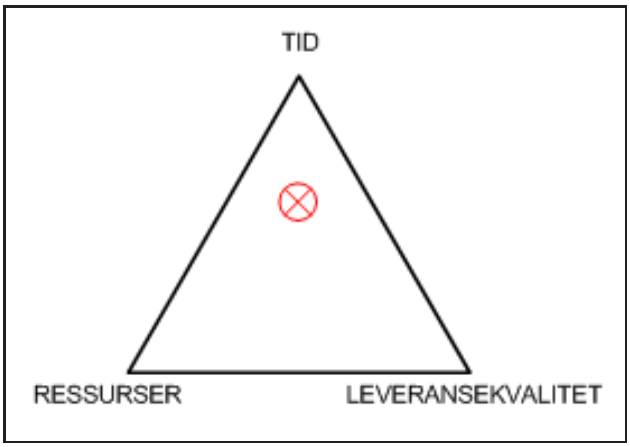
4.4 Nytte-/kostanalyser av prosjekter i SSB

Prosjekter settes i hovedsak igang fordi noen mener prosjektene er lønnsomme, og fordi noen er villige til å betale det det koster å gjennomføre prosjektene. For å kunne evaluere et prosjekt, både underveis og i etterkant, er det viktig å vite hva man ønsker å oppnå gjennom prosjektet, som hvilke utviklingsarbeider som skal gjennomføres, hva gevinstene av prosjektet vil være, og når og hvordan disse skal tas ut.

IT-prosjekter kan beskrives som alle typer prosjekter, investeringer og utviklingstiltak som går ut på å endre bruken av IT i en organisasjon. I organisasjoner hvor alle ansatte så og si til enhver tid er involvert i flere IT-prosjekter er det viktig med et synlig lederskap, fordi det å ta lederskap handler mye om å følge opp og prioritere prosjekter, evaluere og prioritere kompetansen i organisasjonen mellom prosjektene, samt å benytte styringsverktøy for å støtte opp om beslutninger rundt disse temaene. "Systemutviklingsmetoden" (SU-metoden) er et slikt verktøy som definerer en metode for utvikling av programvaresystemer i SSB. I tillegg benytter seksjonsledelsen i seksjon for IT-utvikling seg av verktøy som prosjektskriv, som i grove trekk beskriver formålet med prosjektet og dets beslag på ressurser i virksomheten. Dette er per i dag det dokumentet som danner beslutningsgrunnlaget for nye prosjekter. Avdeling for IT og datafangst har også tatt i bruk balansert målstyring for å kunne følge med på om prosjektenes utvikling er i henhold til planen. Fordelen med at seksjon for IT-utvikling allerede benytter et styringsverktøy som balansert målstyring er at de positive virkningene man ønsker å oppnå for prosjekter for en stor del er definert, og de kan med små justeringen gjenbrukes i en nytte-/kostanalyse.

4.4.1 Tilpasning av en analysemodell

En nytte-/kostanalyse kan brukes av flere parter i en organisasjon. Ledere i organisasjonen kan bruke analysen som et beslutningsgrunnlag for prosjekter, og til prosjektvurderinger i etterkant av prosjekter. Utredere av prosjekter får et arbeidsverktøy til bruk under prosjektplanleggingen og prosjektmedarbeidere lettere kan få økt ansvar i forbindelse med å bidra til bedre styring av prosjektene og øke deres kostnadsbevissthet rundt disse.



Figur 4.2: Kvalitetsperspektiv ved seksjon for IT-utvikling ved SSB

En nytte-/kostnadsmodell må alltid tilpasses den virksomheten hvor evalueringene skal gjøres, fordi kvalitetsperspektivene vil variere fra virksomhet til virksomhet. Kvalitetsperspektivet i seksjon for IT-utvikling i SSB kan illustreres gjennom figur 4.2 hvor trekanten illustrerer de tre viktigste kvalitetsperspektivene i SSB, og krysset hvor ledelsen i seksjon for IT-utvikling har sitt fokus. Figuren illustrerer dermed at det i denne virksomheten først og fremst er viktig at prosjektene holder tidsskjemaet, selv om dette kan kreve at man må fire noe på kravene til kvalitet i systemet og man blir nødt til å omprioritere menneskelige ressurser i virksomheten. I andre virksomheter vil vektingen av kvalitet være annerledes, og man vil kanskje mene at leveransekvalitet er viktigere enn å levere i tide og i forhold til forhåndsestimerte ressurser.

Når jeg her foreslår en nytte-/kostmodell til bruk i SSBs seksjon for IT-utvikling har jeg i første rekke tatt utgangspunkt i CBAM, en modell for nytte-/kostanalyser beskrevet på side 78, og metoden beskrevet av Lien & Bjørn (1990). En nytte-/kostmodell er et verktøy for å evaluere nye og eksisterende prosjekter opp mot de effektmål man har i virksomheten. På den måten er det også klart at en nytte-/kostmodell i seksjon for IT-utvikling i SSB ikke uten videre kan brukes i en annen virksomhet i SSB, fordi de virkningene man velger å måle på i en slik modell vil variere fra virksomhet til virksomhet.

4.4.2 Forutsetninger

Et viktig stikkord når man snakker om nytte-/kostanalyser er forutsetninger. Forutsetningene man setter for en analyse er alfa og omega, fordi verken analyser eller beregninger blir bedre enn de forutsetningene de er bygget på. Forutsetninger kan være prissetting av dagsverk[2], hvor langt tidsperspektiv man skal ha på analysen, når man skal ta ut gevinster, hvor lenge

[2]Dagsverk beregnes gjerne slik

$$\text{Dagsverk} = \frac{\text{årslønn} * 2}{325} \tag{4.4}$$

man regner med at en virkning av et prosjekt varer, hvilken rentefot man skal velge for nåverdiberegningene. Når man snakker om prosjektkostnader og investeringer er det også viktig å huske at man kan dele opp investeringer og spre de over flere prosjekter. På denne måten kan man enklere utjevne utgiftene for prosjektene i virksomheten, og man unngår store skjevheter i kostnadsberegningene for prosjektene.

4.5 Anbefalt analysemodell i SSB

Under vil jeg beskrive modellen for nytte-/kostanalyser slik jeg vil anbefale et slikt verktøy brukt i SSBs seksjon for IT-utvikling. Modellen består av 8 steg:

1. **Valg av prosjekter som skal analyseres** — man velger først de alternativene man skal evaluere opp mot hverandre. Det er viktig å huske at dagens situasjon, dvs nullalternativet, alltid vil være ett av prosjektene

2. **Valg av forutsetninger** — man velger forutsetningene for prosjektet/prosjektene man skal gjøre analysen over

3. **Kvantifiserbare kostnader** — man setter verdier i kroner og øre på hver kvantifiserbare kostnad i modellen for hele tidsperspektivet

4. **Ikke-kvantifiserbare kostnader** — man gir hver ikke-kvantifiserbare kostnad en verdi fra 1 til 3 på hvert av attributtene "sannsynlighet"[3] og "viktighet"[4], og multipliserer disse for å finne totalkostnaden

5. **Kvantifiserbare nytteeffekter** — man setter verdi i kroner og øre på hver kvantifiserbare nytteeffekt i modellen for hele tidsperspektivet

6. **Ikke-kvantifiserbare nytteeffekter** — man gir hver ikke-kvantifiserbare kostnad en verdi fra 1 til 3 på hvert av attributtene "sannsynlighet" og "viktighet", og multipliserer disse for å finne totalgevinsten

7. **Beregne nåverdi** — man velger en kalkulasjonsrente[5] for hele tidsperioden og beregner nåverdien til alternativene ved å diskontere alle beløp. At netto nåverdi beregnes til 0 betyr at pengene like gjerne kan investeres et annet sted, forutsatt at renten er den samme.

8. **Beregne nytte-/kostnadsbrøken** — man beregner summen av diskonterte innbetalinger dividert på summen av diskonterte utbetalinger for hvert av alternativene i analysen. Dersom denne er > 1 har man god grunn til å tro at prosjektet er lønnsomt.

hvor årslønn er en gjennomsnittlig årslønn for virksomheten, man multipliserer denne med to for å dekke opp for sosiale kostnader i virksomheten og man dividerer med antallet arbeidsdager i året, 325.

[3] Beskriver sannsynligheten for at virkningen inntreffer

[4] Beskriver viktigheten av at virkningen inntreffer

[5] Finansdepartementet R14/99 mener at renten i offentlige prosjekter bør ligge mellom fire og åtte prosent

4.5.1 Nytte-/kostmodell

I utgangspunktet vil man ikke identifisere virkninger i en nytte-/kostmodell før prosjektene man ønsker å bruke modellen på er valgt, og forutsetningene for analysen er klar. Når jeg i tabellene for kostnader og nytteeffekter likevel har fylt inn noen virkninger og markert disse enten som kvantifiserbare eller ikke-kvantifiserbare virkninger, er dette fordi jeg mener at disse er så generelle at man ønsker å behandle de som en virkning i nær sagt et hvert IT-utviklingsprosjekt i seksjon for IT-utvikling i SSB. Det vil for eksempel alltid være personalkostnader knyttet til en IT-utviklingsprosjekt, samtidig som kostnader til eksterne datasentraltjenester ikke er en aktuell problemstilling p.t., men er en virkning som kan bli interessant på sikt ettersom bruken av slike tjenester øker både i privat og offentlig sektor. Når det gjelder nytteverdiene er som sagt disse i stor grad utledet fra avdelig for IT og datafangst i SSB sin modell for balansert målstyring, og er dermed aktuelle for nærmest alle IT-utviklingsprosjekter knyttet til seksjon for IT-utvikling ved SSB. Dette kommer av at balansert målstyring i stor grad uttrykker mål knyttet til virksomhetens utviklingsoppgaver. Som et eksempel vil det alltid være interessant å estimere antall feil i systemet eller om systemet kan føre til mindre bruk av overtid i avdelingen. Man bør likevel forsøke å finne virkninger som er enestående for hver enkelt prosjekt man skal analysere.

Under følger en beskrivelse av hver av kostnadsparameterne i figur 4.3 på neste side

1. Alle personalkostnader knyttet til nødvendig opplæring av ansatte i forbindelse med prosjektet

2. Alle personalkostnader knyttet til utviklingsoppgaver i prosjektet

3. Alle personalkostnader knyttet til drifting av den endelige leveransen eller deler av denne

4. Alle personalkostnader knyttet til vedlikehold av den endelige leveransen eller deler av denne

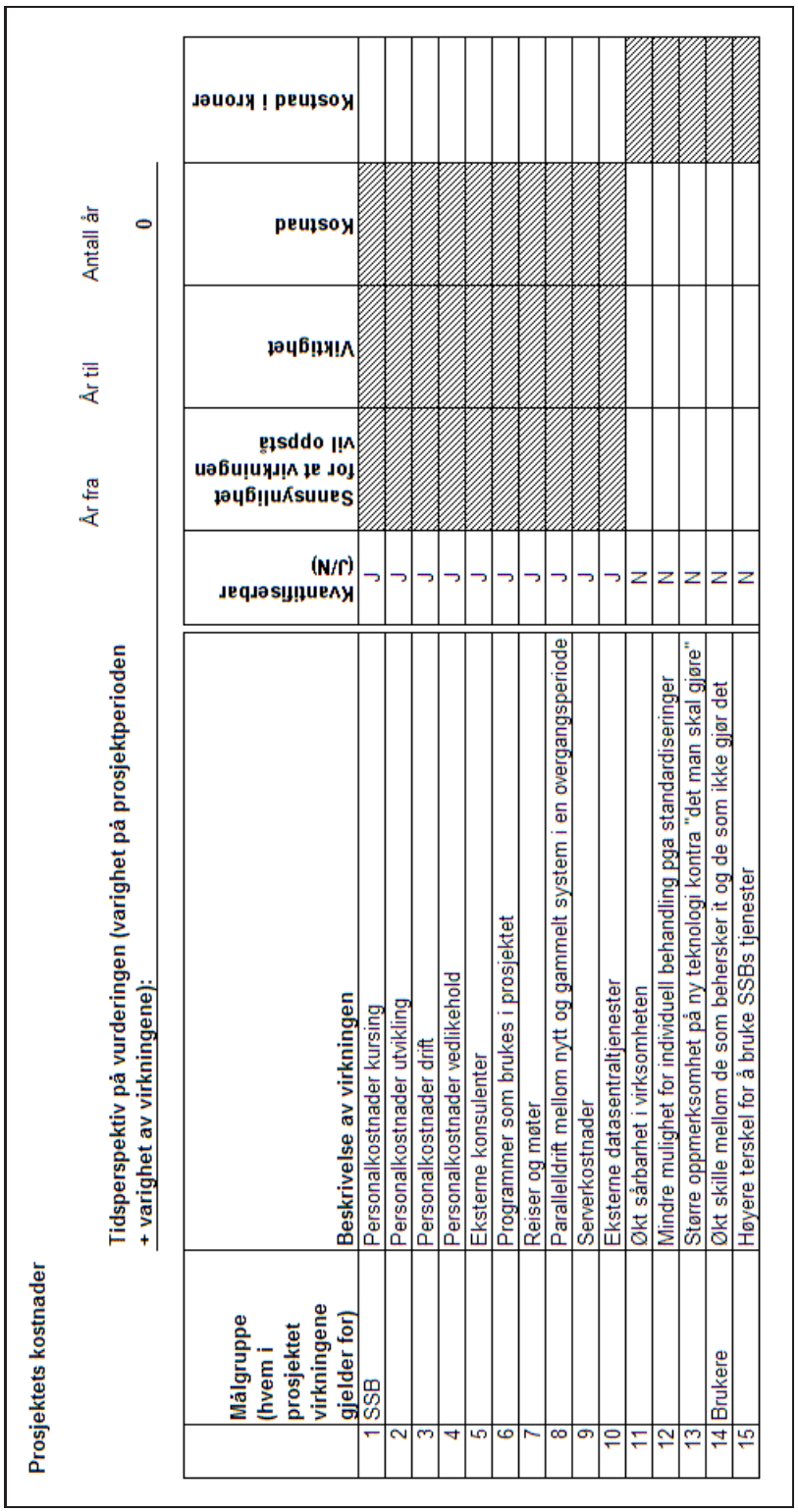
5. Kostnader knyttet til bruk av eksterne konsulenter i alle faser av prosjektet

6. Kostnader knyttet til bruk og/eller anskaffelse av all programvare i prosjektet, som lisenser og oppdateringer.

7. Kostnader i forbindelse med reise- og møtevirksomhet i prosjektet, f.eks. reiser tur/retur Kongsvinger dersom prosjektdeltakere er er fordelt på begge lokasjoner

8. Merkostnader knyttet til dobbeltarbeid i forbindelse med parallelldrift av nytt og gammelt system i en overgangsperiode

9. Merkostnader for servere og datalagre knyttet til mindre god utnyttelse av arkitektur og lagdeling



Prosjektets kostnader

Tidsperspektiv på vurderingen (varighet på prosjektperioden + varighet av virkningene): 0 Antall år

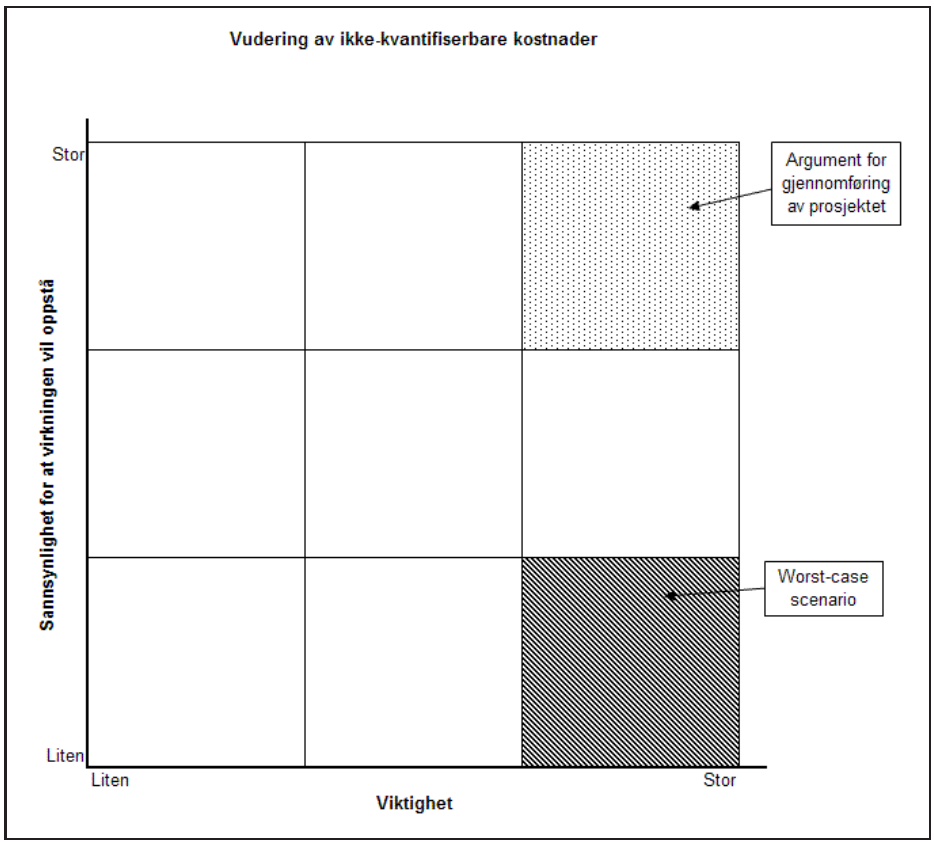
År fra År til

	Målgruppe (hvem i prosjektet virkningene gjelder for)	Beskrivelse av virkningen	Kvantifiserbar (J/N)	Sannsynlighet for at virkningen vil oppstå	Viktighet	Kostnad	Kostnad i kroner
1	SSB	Personalkostnader kursing	J				
2		Personalkostnader utvikling	J				
3		Personalkostnader drift	J				
4		Personalkostnader vedlikehold	J				
5		Eksterne konsulenter	J				
6		Programmer som brukes i prosjektet	J				
7		Reiser og møter	J				
8		Parallelldrift mellom nytt og gammelt system i en overgangsperiode	J				
9		Serverkostnader	J				
10		Eksterne datasentraltjenester	J				
11		Økt sårbarhet i virksomheten	N				
12		Mindre mulighet for individuell behandling pga standardiseringer	N				
13		Større oppmerksomhet på ny teknologi kontra "det man skal gjøre"	N				
14	Brukere	Økt skille mellom de som behersker it og de som ikke gjør det	N				
15		Høyere terskel for å bruke SSBs tjenester	N				

Figur 4.3: Kvantifiserbare kostnader

10. Kostnader knyttet til bruk av eksterne datasentraltjenester i forbindelse med for eksempel outsourcing
11. Kostnader som følge av mer usikre tilkobliger til virksomhetens data, økt risiko for tyverier av personopplysninger og andre data
12. Kostnader som følge av at individuell behandling av brukere blir vanskeligere som følge av standardiserte prosesser og arbeidsrutiner
13. Kostnader som følge av at ansatte bruker mer tid på å teste ny teknologi enn på egne arbeidsoppgaver i prosjektet
14. Kostnader som følge av store sprang i teknologi og arbeidsmønstre som igjen kan føre til større skiller mellom brukere som behersker IT-verktøy godt og de som ikke gjør det
15. Kostnader som følge av at terskelen for å bruke produktet blir høyere blant annet på grunn av avanserte brukergrensesnitt

For de ikke-kvantifiserbare virkningene kan man som nevnt gi disse verdiene 1, 2 eller 3 for sannsynlighet og viktighet, og bruke en 3x3 matrise for å rangere disse, se figur 4.4 på neste side. På den måten får man en oversikt over de virkningene man må prioritere og være ekstra påpasselig med i prosjektgjennomføringen. En slik oversikt vil også være med på å gi beslutningstakere et ikke-økonomisk beslutningsgrunnlag.



Figur 4.4: Ikke-kvantifiserbare kostnader

Under følger en beskrivelse av hver av nytteeffektene i figur 4.5 på neste side

1. Informasjonssikkerheten øker som følge av utviklingsprosjektet
2. Serviceinnstillingen overfor interne og eksterne brukere øker
3. Estimeringspresisjonen forbedres og kravene til prosjektet kommuniseres tydeligere som følge av prosjektet
4. Driftsapparatet blir mer pro-aktivt og forutseende, noe som fører til mindre "brannslukking" enn tidligere
5. Prosjektet fører til at virksomheten er med fremtidsrettet og orientert om ny teknologi og utviklingsmetoder
6. Prosjektet fordrer tilbakeføring av erfaringer, økt kunnskapsdeling og bedre kunnskapsforvaltning i virksomheten
7. Prosjektutviklingen knytter bånd til andre nettverk og oppfordrer til faglig nettverksbygging
8. Prosjektet fører til at systemer får bedre grensesnitt som blant annet kan føre til at man bruker mindre tid på å navigere i dårlige brukergrensesnitt
9. Man bruker mindre tid på feilbehandling av brukermeldte feil som følge av prosjektet
10. Man bruker mindre tid på feilbehandling av systemmeldte feil som følge av prosjektet
11. Færre driftsavbrudd fører til mindre tid brukt på å få systemer opp igjen, slik at tiden kan brukes på andre arbeidsoppgaver
12. Bruk av overtidsarbeid reduseres i forhold til estimert tidsbruk, og denne tiden kan brukes på andre arbeidsoppgaver
13. Andelen sykefravær går ned som følge av prosjektet, og denne tiden kan brukes på arbeidsoppgaver i virksomheten
14. Prosjektet fører til at de ansatte er mer tilfredse på jobb, noe som kan føre til bedre effektivitet, engasjement og bedre ytelse i jobbsammenheng
15. Prosjektet fører til bedre brukertilfredshet og bedre innstilling til SSB, noe som igjen kan føre til bedre kvalitet på innrapporterte data
16. Prosjektet fører til klarere ansvarsforhold og bedre oppdeling av arbeidsprosesser
17. Prosjektet fører til økt automatisering av forretningsprosesser, og tiden man tidligere brukte på å utføre disse prosessene manuelt kan benyttes i andre arbeidsoppgaver

Prosjektets nytteverdier

Tidsperspektiv på vurderingen (varighet på prosjektperioden + varighet av virkningene): 0 År fra År til Antall år

	Målgruppe (hvem i prosjektet virkningene gjelder for)	Beskrivelse av virkningen	Kvantifiserbar (J/N)	Sannsynlighet for at virkningen vil oppstå	Viktighet	Nytteverdi	Nytteverdi i kroner
1	SSB	Styrke sikkerheten i informasjonsbehandlingen	N				
2		Forbedre serviceinnstillingen	N				
3		Klarere og mer realistiske avtaler	J				
4		Økt pro-aktivitet i driftsapparatet	J				
5		Forbedre evnen til nyorientering	N				
6		Tilbakeføre erfaringer	J				
7		Styrke faglig nettverksbygging	N				
8		Bedre brukervennlighet i systemer og datafangstløsninger	N				
9		Færre brukermeldte feil	J				
10		Færre systemmeldte feil	J				
11		Færre driftsavbrudd	J				
12		Redusert avvik mellom planlagte og utførte timeverk	N				
13		Reduksjon i sykefravær	N				
14		Bedre medarbeidertilfredshet	N				
15		Bedre brukertilfredshet	N				
16		Bedre prosessflyt	N				
17		Økt automatisering	J				
18		Reduserte vedlikeholdskostnader	J				
19		Bedre oppfølging av brukere	N				
20		Enklere informasjonstilgang	J				

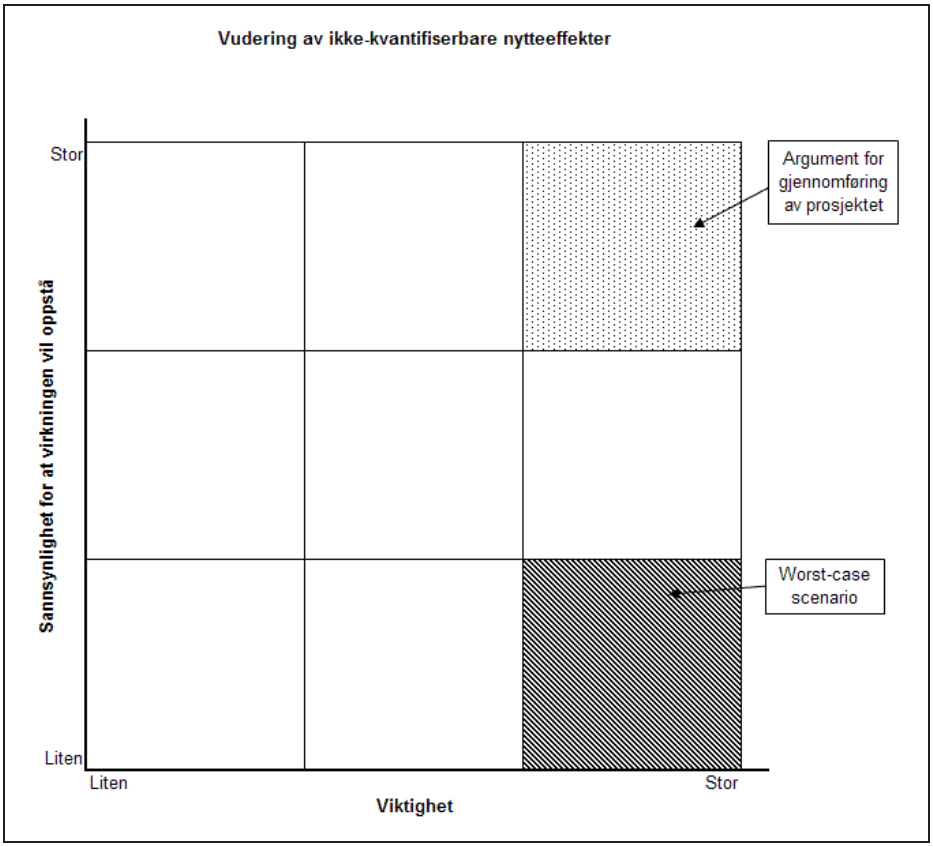
Figur 4.5: Kvantifiserbare nytteeffekter

18. Prosjektet fører til at kostnadene i forbindelse med vedlikehold reduseres og tiden man tidligere brukte på dette brukes på andre arbeidsoppgaver

19. Prosjektet fører til bedre oppfølging av brukere som igjen kan gi bedre brukertilfredshet

20. Prosjektet fører til enklere informasjonstilgang, noe som igjen fører til at man kan bruke den tiden man før brukte på å lete etter riktig informasjon på andre arbeidsoppgaver

For de ikke-kvantifiserbare nytteeffektene kan man som nevnt gi disse verdiene 1, 2 eller 3 for sannsynlighet og viktighet, og bruke en 3x3 matrise for å rangere disse, se figur 4.6.



Figur 4.6: Ikke-kvantifiserbare nytteeffekter

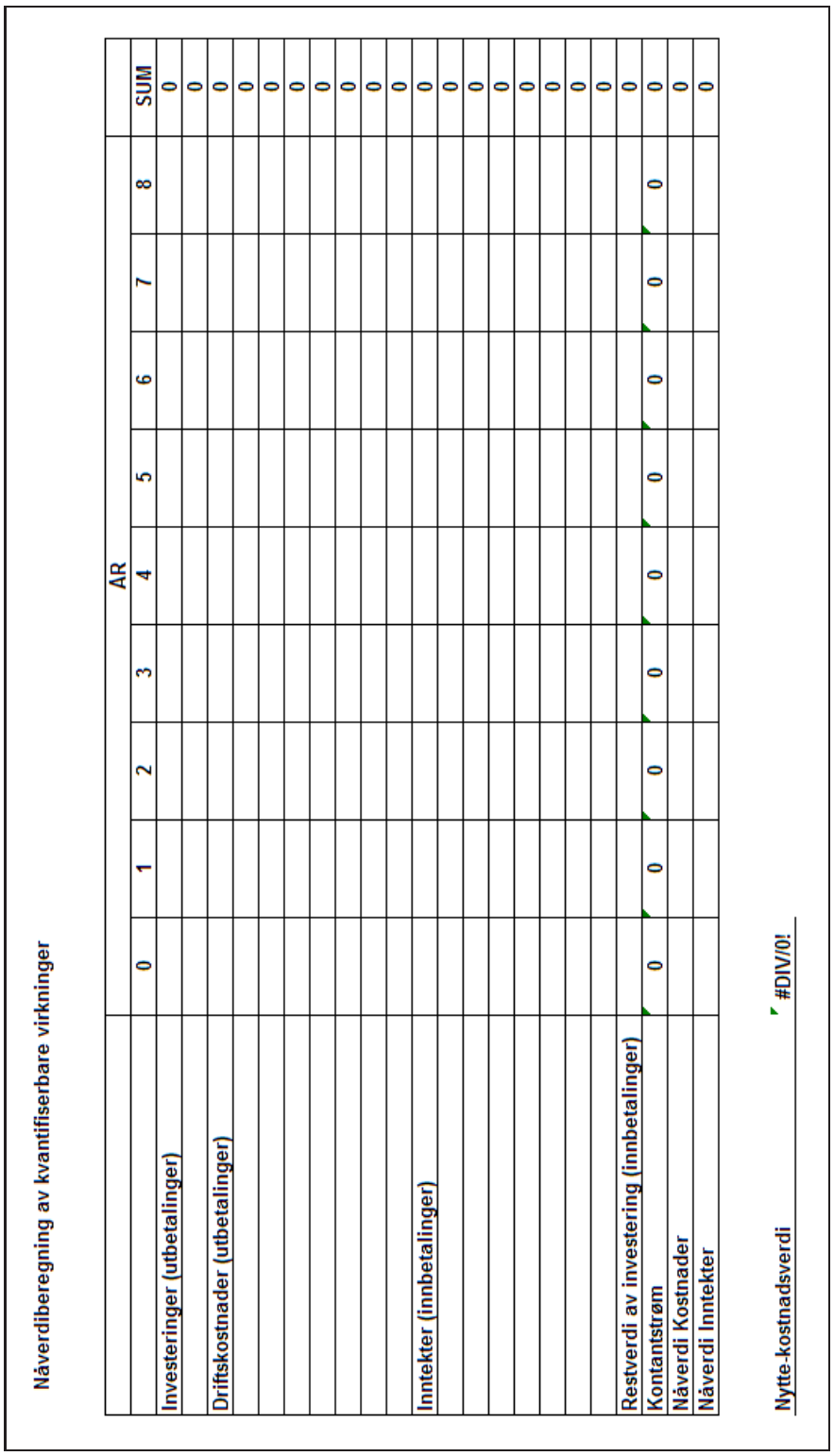
En problemstilling knyttet til de kvantifiserbare virkningene i nytte-/kostmodellen er hvorvidt det er interessant å kunne bruke kroner og øre som benevnelse, slik man normalt vil gjøre i en nytte-/kostanalyse. Grunnen til at en måling i kroner og øre ikke er like interessant for SSB er at de, fordi de er underlagt staten og Finansdepartementet, får sin hovedinntekt gjennom statsoppdraget, som vil være mer eller mindre fast fra år til år. I en slik virksomhet vil det derfor være hvordan man forvalter de midlene man har til rådighet som vil være interessant, og ikke

nødvendigvis hva et utviklingsprosjekt vil koste virksomheten i kroner og øre. Sagt på en annen måte vil det kanskje, for en virksomhet som seksjon for IT-utvikling i SSB, være mer naturlig å måle sine kvantifiserbare virkninger i antall timeverk, da denne benevnelsen er beskrivende for hvordan SSB anvender sine ressurser. For de kvantifiserbare nytteeffektene vil det dermed bli interessant å se hvor mange timeverk man kan omprioritere og bruke på andre oppgaver for hvert enkelt prosjekt.

Når man ikke lenger skal gjøre nytte-/kostanalysen med kroner og øre som benevnelse vil naturlig nok noen av utregningene i analysen bli mindre viktige å gjennomføre. En netto-nåverdiberegning vil for eksempel bli overflødig i en slik virksomhet, fordi man i en slik beregning ville måtte regne om alle timeverk til kroner og øre, diskontere disse opp eller ned til riktig nivå, avhengig av hvilket år man skal beregne timeverksprisen for og hvilket år man skal diskontere til, før man hadde kunnet gjøre selve nåverdiberegningen og fått en verdi som i utgangspunktet ikke er interessant for virksomheten. Isteden vil man summere opp det estimerte antallet timeverk for utbetalinger og innbetalinger, både år for år og totalt, og la dette være de respektive nåverdiene. Dermed kan man likevel regne ut nytte-/kostnadsverdien for et prosjekt ved å dividere timeverk spart på timeverk brukt. Av samme grunn som nevnt over, vil heller ikke utregning av internrenten være av spesiell interesse, da denne renten kun viser hvilken diskonteringsrente som ville fått nytte-/kostverdien til å bli nøyaktig 1.

Under følger en forklaring av noen av begrepene brukt i skjemaet for nåverdiberegning i figur 4.7 på neste side.

- Kontantstrøm — inntektene i prosjektet minus utbetalingene
- Netto nåverdi — diskontering av alle verdier til nåverdi ved hjelp av kalkulasjonsrenten som er valgt. Det er viktig å ta hensyn til at kostnader og nytteeffekter gjerne kommer i fremtid
- Nåverdi Kostnader — netto nåverdiberegning utelukkende for kostnadene
- Nåverdi Inntekter — netto nåverdiberegning utelukkende for inntekter
- Nytte-kostnadsverdi — summen av alle diskonterte inntekter dividert på summen av alle diskonterte utbetalinger. Dersom nytte-kostnadsverdien er større enn 1 er prosjektet per definisjon levedyktig



Figur 4.7: Nåverdiberegning

4.6 Fordeler med nytte-/kostanalyser i SSB

Som jeg var inne på er det tekstlige prosjektskriv som danner beslutningsgrunnlaget for IT-utviklingsprosjekter i SSB i dag, og et åpenbart argument for å ta i bruk nytte-/kostanalyser er at resultatene av disse, dersom forutsetningene er like, i større grad er direkte sammenliknbare sett i forhold til tekstlige beskrivelser av behov, fordeler og ressursbruk av prosjekter. Selv om utformingen av slike prosjektskriv til en viss grad er fast, vil en nytte-/kostanalyse standardisere evaluering av prosjekter på en annen måte, både fordi kostnader og nytteeffekter i langt større grad kvantifiseres og gjøres sammenliknbare, og fordi det ikke på samme måte er rom for tekstlige, subjektive argumenter for å gjennomføre prosjektet.

En nytte-/kostanalyse vil også kunne involvere flere parter i et prosjekt, både med tanke på analysefasen hvor alle virkningene av prosjektet gis verdier, men også i forbindelse med gjennomføringen av selve prosjektet, hvor nytte-/kostanalysen kan brukes som et revisjonsverktøy for å følge opp eksisterende prosjekter. På den måten vil analysen være med på å bevisstgjøre prosjektdeltakere på de kostnader og nytteverdier man forventer å oppnå i prosjektet.

I tillegg til at en nytte-/kostanalyse vil øke kostnadsbevisstheten rundt IT-utviklingsprosjekter vil en slik analyse være et hjelpemiddel for å hente ut gevinstene som oppstår som følge av prosjektet. Med dette mener jeg at fordi en slik analyse beskriver kostnader og nytteeffekter i et prosjekt år for år gjennom hele prosjektets levetid vil analysen være et verktøy i arbeidet med å bevisstgjøre virksomheten på hvilke gevinster som kan hentes ut på hvilket tidspunkt i et prosjekt. Dette gjør også at man enklere kan bestemme hvordan disse gevinstene skal realiseres.

Kostnadsbevissthet i offentlig sektor vil av og til være en utfordring fordi organisasjoner som SSB ikke nødvendigvis er interessert i hva prosjekter koste i kroner og øre. Organisasjoner i offentlig sektor har likevel ikke ubegrensede midler, og enheter som timeverk, dagsverk og årsverk vil være aktuelle størrelser for å måle ressurser i organisasjonen. Det vil uansett, offentlig eller privat sektor, være viktig å utnytte de ressursene man har på beste mulige måte, noe en bevisstgjøring gjennom nytte-/kostanalyser kan være et hjelpemiddel for.

Kapittel 5

Konklusjon

I denne masteroppgaven og i dette dokumentet har jeg sett på og designet en mulig referansearkitektur for en integrasjon av rapporteringssystemene Kostra og Idun i SSB, i kontekst av tjenesteorienterte prinsipp. Jeg har også utformet en nytte-/kostmodell for evalueringer av IT-utviklingsprosjekter i offentlig sektor.

5.1 Evaluering av IT-utviklingsprosjekter

Jeg har tidligere argumentert for viktigheten av å evaluere IT-utviklingsprosjekter for å måle om gevinstene man kan ta ut av et prosjekt er større enn kostnadene, og at prosjektet således er lønnsomt per definisjon. En slik evaluering er også viktig for å konkretisere gevinstene og dermed vite når og hvordan disse skal tas ut. Innføring av et verktøy som en nytte-/kostanalyse vil kreve økt bevisstgjøring av ansatte og deres arbeid i utviklingsarbeidet, og kanskje også kreve at man ser IT-utviklingsprosjekter på en ny måte. Man må være klar over at en slik bevisstgjøring ikke nødvendigvis vil bli enkel å oppnå, da det krever en endring i de ansattes holdninger og syn på eget arbeid og egne arbeidsprosesser, samt virksomhetsprosessene og måten man utvikler IT-systemer på. Nytte-/kostanalyser er også i stor grad et styringsverktøy for kostnader og gevinster i prosjekter. Man vil, gjennom bruk av et slik verktøy, ha god oversikt gjennom hele prosjektperioden over hvilke og hvor store kostnader man har estimert med, og når og hvordan gevinstene som kommer som følge av prosjektet skal tas ut. På den måten blir det enklere å begrunne eventuelle omprioriteringer av ressurser i prosjektet dersom man underveis ser at avvikene mellom estimat og faktiske kostnader eller gevinster er større enn en gitt toleransegrense. I tillegg vil slik analyser være et verktøy for å måle nytteverdiene i et fremtidig prosjekt, da tidligere analyser og evalueringer er gode historiske data som kan være med og danne grunnlaget for estimat og målinger i nye prosjekt.

Som jeg påpeker i Kapittel 4 fra side 69 er man helt avhengig av gode måleparametere for at en analyse skal bli så nøyaktig som mulig. Selv om enkelte parametere er listet opp i modellen for SSB, er det viktig å bruke tid på disse før en evaluering, da parameterne man ønsker å

måle på gjerne varierer noe fra prosjekt til prosjekt. I tillegg til kart definerte måleparametere er forutsetningene for analysene alfa og omega, da en analyse aldri blir bedre enn forutsetningene den er basert på. Ved å bruke et slikt verktøy vil det imidlertid bli enklere å hente ut gevinster fra et prosjekt, da disse spesifiseres i evalueringen, samtidig som estimeringspresisjonen og kostnadsbevisstheten i virksomheten vil kunne øke på bakgrunn av historiske data og evaluering av analysene etter prosjektets slutt.

5.2 Referansearkitektur

Mitt forslag til referansearkitektur for en integrert løsning av rapporteringssystemene Kostra og Idun, beskrevet i Kapittel 3 fra side 35 og i Tillegg B fra side 117, er å regne som en plattformuavhengig referansearkitektur for systemintegrasjonen, i henhold til COMET, hvor jeg ikke har tatt noen teknologiske valg for det endelige systemet, utover det faktum at arkitekturen er bygget på tjenesteorienterte prinsipp. Slike valg vil det være naturlig og nødvendig å gjøre ved en overgang fra en plattform uavhengig modell (Platform Independent Model, PIM) til en plattform spesifikk modell (Platform Specific Model, PSM). Jeg har likevel kort beskrevet ulike tilnærminger til en realisering av en tjenesteorientert arkitektur i Kapittel 2 fra side 11 og mener at en tjenesteorientert arkitektur med Web Services vil være et aktuelt alternativ for integrasjonsløsningen av Kostra og Idun. Dette vil jeg blant annet begrunne med at en slik løsning med Web services i størst grad vil oppfylle kravene til en tjenesteorientert arkitektur. En tjenesteorientert arkitektur med Web services er basert på XML, som er en åpen standard. Arkitekturen vil ha støtte for flere ulike programmeringsspråk, løse kobliger, autentisering og autorisasjon, i tillegg til støtte for flere ulike kommunikasjonsformer. Den kanskje største fordelen med Web Services er likevel at plattformen er komponentbasert.

5.2.1 Komponentbasert utvikling

Komponentbasert systemutvikling og å skulle implementere systemet i flere steg kan også være en fordel i andre sammenhenger, blant annet fordi man av erfaring vet at det vil kreve mye arbeid å opprette et fullstendig tjenestebibliotek for hele rapporteringssystemet, og hvor alle tjenestene som brukes er inneholdt. I tillegg må man være klar over at det er det store kostnader knyttet til en implementasjon av en tjenesteorientert arkitektur. En slik integrasjon krever endringer i, om ikke annet spesifikasjon av, nær sagt alle virksomhetsapplikasjoner tilknyttet systemet og applikasjoner som støtter kommunikasjon virksomheter i mellom dersom man ønsker et størst mulig utbytte av arkitekturen. Dette er nødvendig for å oppfylle kravet om løse kobliger mellom tjenestene og systemkomponentene i en slik arkitektur, og fordi applikasjonene må tilpasses nye meldingsmønstre.

5.2.2 Sikkerhet

Det faktum at Web Services plattformen er komponentbasert er en stor fordel med tanke på at et komplett sett av standarder for plattformen ennå ikke foreligger. Spesielt relevant er det at standarder knyttet til ulike sikkerhetsmekanismer ikke er komplette, fordi skjema med sensitive personopplysninger er avhengige av gode sikkerhetsrutiner for at disse skal kunne sendes sikkert over Internett, og brukes og lagres sikkert på maskiner med direkte tilkobling til Internett. Fordi Web Services er en komponentbasert plattform kan man derfor i første omgang utvikle komponenter til systemet som ikke krever de samme sikkerhetsmekanismene som data med sensitive personopplysninger, og dermed bygge opp et fullstendig og komplett system i flere steg og over en lengre tidsperiode enn man normalt ville gjort om man skulle utvikle hele systemet samlet. Jeg var i Kapittel 3 inne på at blant andre Rikstrygdeverket har løst problemet med lagring av sensitive data på maskiner med direkte Internet-tilgang, og oversending av slike data over Internett, ved å benytte en soneinndeling av virksomhetens maskinpark. På den måten vil de ulike sonene gi retningslinjer og restriksjoner i forhold til tillatt behandling, og eventuelt dekryptering, av sensitiv informasjon. Slik informasjon vil man normalt ikke kunne dekryptere og lagre før de er på en maskin i en sikret sone i virksomheten. Avhengig av behovet for å beskytte data vil man kunne dele inn virksomheten i soner med ulike sikkerhetsgrader.

5.2.3 Fremtidig arbeid

I mitt design av arkitektur for systemintegrasjonen har jeg valgt å ta hensyn til mål og forretningsprosesser, for ikke å være avhengig av dagens arbeidsmønster og prosesser i virksomheten, men heller stå fritt i forhold til å skape nye prosesser tilpasset målene i virksomheten og de tjenestene systemet skal levere. I Kapittel 3 fra side 35 nevnte jeg at skjemagenereringsverktøyene for Kostra og Idun ikke er tilfredsstillende, og at det ved en overgang til en tjenesteorientert arkitektur vil være nødvendig å implementere et nytt skjemagenereringsverktøy tilpasset en tjenesteorientert arkitektur. Det vil være viktig at et slikt verktøy er fleksibelt nok til å imøtekomme krav både fra Kostra og Idun om godt skjemadesign, at verktøyet genererer utskriftsvennlige skjema til bruk i en online portal og at det støtter publisering av ferdig utviklede skjema til portalløsningen.

5.2.3.1 Revisjonssystem

SSB har i dag flere revisjonssystem som brukes for å revidere skjema i Kostra, og alle disse har hver sine spesielle tilpasninger og funksjoner som støtter ønskene fra fagseksjonen som bruker dem. I mitt design av referansearkitektur har jeg bare modellert med ett revisjonssystem. Jeg mener det er lite hensiktsmessig å ha flere tilnærmet like revisjonssystem, blant annet fordi man får flere systemer å vedlikeholde Jeg mener derfor

at man bør vurdere å utvikle ett felles revisjonssystem for alle skjema i SSB generelt og i Kostra spesielt.

5.2.3.2 Portalen

Man må også se på hvordan portalen skal utformes for å støtte pålogging og innsending av skjema både fra offentlige kontorer og institusjoner og fra næringslivet generelt, da oppgavegiverne fra disse to avgivergruppene identifiseres på ulike måter slik systemene er lagt opp i dag. I denne sammenhengen vil det også være viktig å kartlegge Internet-tilgangen i de ulike kommunene og i alle institusjoner og virksomheter som skal bruke rapporteringsløsningene for Kostra og Idun. Man kan ikke utelukkende ta det for gitt at alle institusjoner som barnehager, kirker, barnevernsinstitusjoner og liknende har tilgang til Internett, eller gode nok Internett-forbindelser for at arbeidet med å svare på skjema fra SSB ikke tar unødvendig lang tid på grunn av for liten båndbredde. Det vil samtidig være viktig at skjema i Kostra og Idun er forholdsvis lette å laste opp i en portal, slik at kravene til båndbredden blir så lave som mulig.

5.2.3.3 Risikovurdering

Jeg har valgt å ikke ta hensyn til risiki knyttet til en implementasjon av arkitekturen i Kapittel 3 på side 64, hovedsaklig fordi det ikke foreligger klare teknologivalg for systemet, valg som i seg selv kan påvirke risiki og aktiva i systemet og i virksomheten. Det vil derfor være svært aktuelt å gjennomføre en risikoanalyse når man har tatt alle de nødvendige teknologivalgene. Gjennom en risikoanalyse vil man identifisere alle aktiva i systemet og alle risiki knyttet til disse, sannsynligheten for at de ulike risiki skal inntreffe og hvilken konsekvens dette vil få for hvert aktivum i systemet. Risikoanalyser er spesielt viktig i systemutviklingsprosjekter da det i utgangspunktet er knyttet stor usikkerhet til disse, fordi det er vanskelig å estimere faktisk bruk av tid og ressurser i et prosjekt, i tillegg til at kravene til systemet kan endres over tid.

5.2.3.4 Analyse av systemarkitektur

Før man velger å ta i bruk en ny systemarkitektur, som den jeg forslår i Kapittel 3 på side 64, er det viktig å vite om systemet realisert av arkitekturen kan holdes innenfor gitte økonomiske rammer, både under utvikling og når det står ferdig, samtidig som det gir avkastninger og nytteeffekter som gjør at det vil lønne seg å utvikle dette, fremfor å holde seg til dagens løsning, nullalternativet. For å gjennomføre en slik analyse kan man benytte nytte/kostmodellen som jeg foreslår i Kapittel 4 fra side 69, for å kunne kvantifisere kostnader og gevinster i prosjektene og på den måten danne et kvantitativt beslutningsgrunnlag for hvilke prosjekt som skal igangsettes basert på lønnsomhet. Samtidig vil resultatet av en slik analyse også fungere som et styringsdokument til videre bruk under selve prosjektgjennomføringen.

Helt til slutt er det viktig å påpeke at mitt forslag til referansearkitektur for et integrert system av Kostra og Idun ikke er en endelig systemarkitektur. Med det mener jeg at flere iterasjoner over modellene vil være nødvendig for å modellere systemet og systemarkitekturen på et riktig detaljnivå. I tillegg vil det være fornuftig også å modellere interaksjonen mellom de ulike systemkomponentene for å øke forståelsen av meldingsflyten i systemet.

5.3 Fremtidens systemintegrasjon

Integrasjon er tradisjonelt blitt utført manuelt, hvor man endrer eksisterende programvare slik at denne også fungerer sammen med annen programvare. I tillegg til at manuell integrasjon er dyrt og at det gjerne tar lang tid, er det også vanskelig å gjenta prosessen, samtidig som denne er vanskelig å dokumentere. Libes, Flater, Steves, Feeney & Barkmeyer (2004) skriver at manuell integrasjon ofte blir så kostbart at man i de fleste tilfeller benytter andre alternativ, som å forkaste eksisterende programvare og dermed begynne helt forfra.

Den manuelle integrasjonens rake motsetning er naturligvis automatisk integrasjon. Ideen med automatisk integrasjon er at integrasjonen skal skje helautomatisk og håpet er dermed at man skal kunne spare arbeidskraft. Ideelt sett, skriver Libes et al., er automatisk integrasjon et viktig mål, men det er foreløpig mange skjær i sjøen før man kommer så langt. Det kan likevel være nyttig å strebe etter å nå dette målet, hvor uoverkommelig det enn måtte virke, for på den måten å utvikle metoder for systemintegrasjon til det stadig mer ekstreme.

5.3.1 Standarder

Standarder innen informasjonsteknologi er et særdeles viktig element i automatisk integrasjon. Det er imidlertid viktig å merke seg, som Libes et al. (2004) påpeker, at standarder ikke automatisk løser alle problemer. Standarder kommer ofte i forskjellige versjoner, det er alltid en ny standard under utvikling og man må ofte velge mellom flere motstridende standarder. Eksempler på standarder som er i stadig forandring er overgangen fra SGML til HTML og derfra til XML. Automatiserte metoder er, i følge Libes et al. (2004), blitt utropt som en mulig løsning på dilemmaene rundt ukoordinerte standarder, ontologier, eldre systemer og de stadig økende kostnadene knyttet til manuell integrasjon. Kort sagt håper man at de automatiserte metodene skal redusere de tradisjonelle kostnadene ved manuell systemintegrasjon.

5.3.2 Modelldrevet arkitektur

Modelldrevet arkitektur (MDA), som OMGs MDA, lover på sin side teknologiuavhengig utvikling, ved å flytte fokus fra systemenes særegne egenskaper til et høyere abstraksjonsnivå. Dette prinsippet vil fungere fint for utvikling av nye systemer, men det er vanskelig å si hvilken innflytelse

dette vil få på de eldre systemene som ikke har noen plattformuavhengig modell som en MDA krever, eller en implementasjonsmodell over det ferdige systemet.

Bibliografi

Abelsæth, A. e. (2001), IDUN systemdokumentasjon. Publisert til internt bruk i SSB.

Agrawal, R., Bayardo Jr., R. J., Gruhl, D. & Papadimitriou, S. (2002), 'Vinci: a service-oriented architecture for rapid development of web applications', *Elsevier Science B. V.* .

Alonso, G. & Casati, F. (2005), 'Web services and service-oriented arrchitectures', *ICDE* .

AltInn (n.d.), 'Om altinn', https://www.altinn.no/cms/1044/altinn/Mer+om+altinn/. Besøkt: 16.12.2005.

Arsanjani, A. (2002), 'Developing and integrating enterprise components and services', *Communications of the ACM* .

Arsanjani, A. (2004), 'How to identify, specify and realize servicer for your SOA', *SOA and Web Services Center of Excellence, IBM* .

Arsanjani, A., Hailpern, B., Martin, J. & Tarr, P. (2003), 'Web services promises and compromises', *QUEUE* .

Atkinson, Bob, e. (2002), Web services security (ws-security). Besøkt 18.02.2005.
*http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnglobspec/html/ws-security.asp

Baglietto, P., Maresca, M., Parodi, A. & Zingirian, N. (2004), 'Stepwise deployment methodology of a service oriented architecture for business communities', *Elsevier B. V.* .

Baker, S. & Dobson, S. (2005), 'Comparing service-oriented and distributed object architectures', *Springer-Verlag* .

Balsamo, S., Inverardi, P. & Mangano, C. (1998), 'An approach to performance evaluation of software architectures', *ACM Press* .

Berre, A.-J., Elvesæter, B., Aagedal, J., Oldevik, J., Solberg, A. & Nordmoen, B. (2004), *COMET Methodology Handbook*, SINTEF ICT.

Biggerstaff, T. J. (1998), 'A perspective of generative reuse', *Annals of Software Engineering 5 169-226, Microsoft Research* .

Boehm, B., Brown, J. & Lipow, M. (1976), 'Quantitative evaluation of software quality', *TRW Systems and Energy Group* .

Bosch, J. & Molin, P. (1997), 'Software architecture design: Evaluation and transformation'.

Butler, S. (2002), 'Security attribute evaluation method: A cost-benefit approach', *ICSE* .

Carnegie Mellon University (2006), 'How do you define software architecture', http://www.sei.cmu.edu/architecture/definitions.html.

Chen, D. & Doumeingts, G. (2003), 'European initiatives to develop interoperability of enterprise applications - basic concepts, framework and roadmap', *Elsevier Ltd.* .

Chung, J.-Y. (2005), 'An industry view on service-oriented architecture and web services', *SOSE* .

Chung, S., Bylin, Z. & Davalos, S. (2005), 'Service-oriented development and integration: Toward web services-based business information systems', *ICWS* .

Colan, M. (2002), Service-oriented architecture expands the vision of web services, part 1, characteristics of service-oriented architecture. Besøkt 01.02.2005.
*http://www-106.ibm.com/developerworks/webservices/library/ws-soaintro.html

Collignon, S. & Cook, S. (2002), 'Review of software integration and techniques of cots products in the commercial environment', *Systems Enineering, Test & Evaluation Conference, Sydney, Australia* .

Dale, T. & Hole, B. (2005), Evalueringer av elektroniske skjema i KOSTRA, Technical report, Seksjon for Datafangstmetoder, SSB.

Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M. & Newling, T. (2004), 'Patterns: Service-oriented architecture and web services', *IBM RedBooks* .

Erlanger, L. (2005), Transamerica turns silos into services. Besøkt 03.10.2005.
*http://www.infoworld.com/infoworld/article/05/05/02/18FEsoatransamerica1.html

Fløysvik, T., Eig, T. & Svinnset, A.-B. (2003), KOSTRA (kommune - stat - rapportering), rutinebeskrivelser of dokumentasjon. Publisert til internt bruk i SSB.

Forvaltningsinfo AS (2003), IT mot skjemabelastning, Technical report, Nærings- og handelsdepartementet.

Forvaltningsinfo AS (2005), ELMER 2 - forslag til retningslinjer for brukergrensesnitt i offentlige skjemaer på internett, Technical report, Nærings- og handelsdepartementet.

Gartner Consulting (2004), Rapport om mulige fællesoffentlige arkitekturkrav, Technical report, Ministeriet for Videnskab, Teknologi og Udvikling, Danmark.

Greene, J. (2005), *Applied Software Project Management*, O'Rielly.

Gruman, G. (2005), SOA ensures guardian gets it right. Besøkt 03.10.2005. *http://www.infoworld.com/infoworld/article/05/05/02/18FEsoaguardian_1.html

Heimly, V. & Grøtting, K. A. (2005), 'It-strategi en håndbok for helsesektoren', http://www.kith.no/upload/913/R05-97HandbokIT-strategi.pdf. Kapittel 14.

Hervik, A., Hagen, K. P., Nyborg, K., Scheel, H. H. & Sletner, I.-J. (1997), 'Nytte-kostnadsanalyser - prinsipper for lønnsomhetsvurderinger i offentlig sektor', Norges offentlige utredninger, NOU 1997: 27. Finans- og tolldepartementet.

Hogg, K., Chillcott, P., Nolan, M. & Srinivasan, B. (2004), 'An evaluation of web services in the design of an b2b application', *Australian Computer Society, Inc.* .

Huang, Y., Li, Y. & Chao, K.-M. (2005), 'A framework for service-oriented business integration under uncertainty', *ICEBE* .

IBM (ukjent publiseringsår), New to SOA and web services. Besøkt 01.02.2005. *http://www-106.ibm.com/developerworks/webservices/newto/

IT og Telestyrelsen (2002), Grønbog om IT-arkitektur, Technical report, Ministeriet for Videnskab, Teknologi og Udvikling, Danmark.

Kazman, R., Asundi, J. & Klein, M. (2001), 'Quantifying the costs and benefits of architectural decisions', *IEEE* .

King, J. L. & Schrems, E. L. (1978), 'Cost-benefit analysis in information systems development and operation', *Computing Surveys* **10**(1).

Kommunal- og regionaldepartementet (2004), Rapport - overholdelse av rapporteringsfrister i KOSTRA, Technical report, Kommunal- og regionaldepartementet.

Krafzig, D., Banke, K. & Slama, D. (2005), *Enterprise SOA, Service Oriented ARchitecture Best Practices*, The Coad Series, Prentice Hall PTR.

Král, J. & Žemliča, M. (2004), 'Towards design rationales of software confederations', *ICEIS 2004* .

Larsen, K. & Bloniarz, P. A. (2000), 'A cost and performance model for web service investment', *Communications of the ACM* . Vol. 43, No. 2.

Lee, Y. & Choi, H.-J. (2005), 'Experience of combining qualitative and quantitative analysis methods for evaluating software architecture', *ICIS* .

Libes, D., Flater, D., Steves, M., Feeney, A. B. & Barkmeyer, E. (2004), 'The challenges of automated methods for integrating systems', *National Institute of Standards and Technology, Maryland, USA* .

Lien, B. & Bjørn, Ø. (1990), *Kost-nytteanalyse av IT-prosjekter*, Statskonsult.

Linthicum, D. S. (2004), *Next Generation Application Integration*, Addison-Wesley.

Ma, K. (2005), 'Web services: What's rela ans what's not?', *IEEE* .

Microsoft-Corporation (2001), Global XML web services architecture. Besøkt 18.02.2005.
*http://www.gotdotnet.com/team/XMLwebservices/gxa_overview.aspx

Moderniseringsdepartementet (2005), 'enorge 2009 - det digitale spranget'.

Moitra, D. & Ganesh, J. (2004), 'Web services and flexible business processes: towards the adaptive enterprise', *Elsevier B. V.* .

Newcomer, Eric, G. L. (2004), *Understanding SOA with Web Services*, Addison-Wesley.

Nielsen, H. F., Christensen, E., Lucco, S. & Levin, D. (2001), Web services referral protocol (WS-referral). Besøkt 18.02.2005.
*http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnglobspec/html/ws-referral.asp

Nielsen, H. F. & Thatte, S. (2001), Web services routing protocol (WS-routing). Besøkt 18.02.2005.
*http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnglobspec/html/ws-routing.asp

Niven, P. R. (2003), *Balanced Scorecard Step-by-step for Government and Nonprofit Agencies*, John Wiley & Sons, Inc.

Patrick, P. (2005), 'Impact of soa on enterprise information architectures', *BEA Systems, Inc.* .

Ravichandran, T, M. A. R. (2003), 'Software reuse strategies and component markets', *Communications of the ACM* .

Remenyi, D., Money, A., Sherwood-Smith, M. & Irani, Z. (2000), *The Effective Measurement and Management of IT Costs and Benefits, 2nd edition*, Butterworth-Heinemann.

Röder, H.-P. & Flikke, T. (2005), 'Slik får du mer igjen for it-pengene', http://www.steria.no/?id=1140. Besøkt: 14.02.2006.

Sand, P., Holm, T., Pedersen, L., Moaffi, H., Dale, T., Folkedal, J., Hole, B. & Krogsrud, A. (2006), Kostra arbeidsgruppe. Publisert til internt bruk i SSB.

Sandvik, P. (2002), Mottak av KOSTRA data i SSB. Publisert til internt bruk i SSB.

SAP (2003), 'SAP netweaver solution beief', *SAP* .

Sassone, P. G. (1988), 'Cost benefit analysis of information systems: A survey of methodologies', *ACM* . College of Managemnet, The Georgia Institute of Technology.

Slaughter, S., Harter, D. E. & Krishnan, M. S. (1998), 'Evaluating the cost of software quality', *Communications of the ACM* .

Sommerville, I. (2001), *Software Engineering*, Addison Wesley, chapter 23.2 Estimation techniques.

Specht, T., Drawehn, J., Thränert, M. & Kühne, S. (2005), 'Modeling cooperative business processes and transformation to a service oriented architecture', *CEC* .

Statistisk sentralbyrå (1999), 'Statistikkloven', http://www.ssb.no/omssb/statlov. Besøkt: 04.05.2005.

Statistisk sentralbyrå (2002), 'Strategiplan 2002 for statistisk sentralbyrå', http://www.ssb.no/omssb. Besøkt: 05.05.2005.

Statistisk sentralbyrå (2004), 'Dette er statistisk sentralbyrå, en institusjon som teller', *Statistisk sentralbyrå* . Besøkt: 04.05.2005. *http://www.ssb.no/omssb/dette_er_ssb.pdf

Statistisk sentralbyrå (n.d.*a*), Idun systemarkitektur. Publisert til internt bruk i SSB.

Statistisk sentralbyrå (n.d.*b*), 'Virksomhet og planer 2004-2005', http://www.ssb.no/omssb/2005-6.pdf. Besøkt: 04.05.2005.

Sundvoll, A. (2005), Kirkelig tjenestestatistikk i KOSTRA-drakt. et pilotprosjekt., Technical report, Seksjon for datafangstmetoder, SSB.

Tockey, S. (2004), *Return on Software: Maximizing the Return on Your Software Investment*, Addison Wesley Professional.

Ukjent forfatter (2002), Commentary - where did they go, i just don't know... Besøkt 14.02.2005.
*http://www.cbdiforum.com/public/news/index.php3?id=885

Vinoski, S. (2003), 'Integration with web services', *IEEE Internet Computing* .

Watkins, T. (2003), Introduction to cost benefit analysis. Besøkt 16.08.2005.
*http://www2.sjsu.edu/faculty/watkins/cba.htm

Wilkes, L. (ukjent publiseringsår), ROI - the costs and benefits of web services and service oriented architecture. Besøkt 14.02.2005.
*http://roadmap.cbdiforum.com/reports/roi/

Wolter, R. (2001), XML web services basics. Besøkt 18.02.2005.
*http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnwebsrv/html/webservbasics.asp

Zhu, L., Babar, M. & Jeffery, R. (2004), 'Mining patterns to support software architecture evaluation', *WICSA* .

Tillegg A

Realisering av tjenesteorienterte arkitekturer

Tjenesteorientert arkitektur har i den senere tid spilt en viktig rolle i mange ulike IT-prosjekter. Dette kommer av at det å bruke tjenesteorientert arkitektur er en måte å designe systemer på som tar alle aspekter ved å utvikle og bruke virksomhetstjenester i øyesyn, som modellering, design, utvikling, bruk, drift, versjonshåndtering og avvikling. Det er imidlertid viktig å være klar over at en investering i en tjenesteorientert arkitektur er en investering inn i fremtiden. Med dette menes at de reelle avkastningene gjerne lar vente på seg, og at fordelene med en tjenesteorientert arkitektur sjelden er synlige før arkitekturen har vært i bruk en viss tid. For å unngå høye utgifter og lave avkastninger til å begynne med, kan det være lurt å begynne i det små, og gradvis bygge opp en fullverdig tjenesteorientert arkitektur over tid.

En tjenesteorientert arkitektur kan implementeres med flere ulike teknologier og produkter (Ma 2005) som CORBA, J2EE eller ferdigproduserte implementasjoner. En av teknologiene som er blitt mye diskutert i det siste er Web services, en kombinasjon av konsepter i web- og tjenesteorienterte arkitekturer (Specht, Drawehn, Thränert & Kühne 2005).

A.1 WebSphere MQ

Mange store organisasjoner har utviklet tjenesteorienterte arkitekturer ved å bruke WebSphere MQ fra IBM, blant annet AXA Financial (Newcomer 2004). I og med at WebSphere MQ ganske enkelt er meldingsorientert mellomvare, må brukeren av WebSphere MQ selv utvikle tjenesteplattformen, som blant annet inkluderer en spesifikasjon for hvordan kontrakter skal defineres, en definisjon av meldingsformatet, definisjoner av hvordan tjenester skal registreres og oppdages, sikkerhetsmekanismer på tjenestenivå og generell håndtering av tjenestene (Newcomer 2004).

En av grunnene til at enkelte likevel velger å bruke WebSphere MQ kan være at tradisjonene for å bruke IBM-løsninger i organisasjonen er lange, og at de i tillegg til å bruke WebSphere MQ også bruker andre teknologier for å støtte opp om de deler av den tjenesteorienterte arkitekturen som

WebSphere MQ i utgangspunktet ikke har noen støtte for. WebSphere MQ går dessuten for å være enkelt å lære, da API'en til WebSphere MQ er relativt liten (Newcomer 2004). De viktigste argumentene for å velge WebSphere MQ som en teknologi for å utvikle en tjenesteorientert arkitektur må likevel være at WebSphere MQ i stor grad fremmer løse koblinger mellom de ulike tjenestene, noe som anses for å være et viktig prinsipp i en tjenesteorientert arkitektur, i tillegg til at WebSphere MQ garanterer levering av meldinger. I mange sammenhenger er sistnevnte et av de viktigste kravene til en leveringskritisk applikasjon som bruker tjenesteorientert arkitektur (Newcomer 2004).

A.2 SAP NetWeaver

SAP NetWeaver tilbyr verktøy og tjenester som i stor grad forutsetter teamarbeid på tvers av forretningsprosesser (SAP 2003). SAP NetWeaver skal også gjøre det enklere for de ulike partene i prosessen å dele kunnskap, applikasjoner og ideer i real-time. Som et resultat av dette mener SAP at man vil bryte ned barrierer mellom mennesker, informasjon og resultater.

SAP NetWeaver er en åpen og omfattende integrasjons- og applikasjonsplattform, som ifølge SAP (2003) skal integrere mennesker, informasjon og forretningsprosesser, og på den måten redusere organisasjonens totale kostnader. SAP NetWeaver kan også kommunisere med miljøene rundt Microsoft .NET og IBM WebSphere MQ. SAP NetWeaver muliggjør samtidig Enterprise Service Architecture, en plan for en komplett tjenesteorientert virksomhetsløsning som kombinerer kraften til virksomhetsapplikasjonene med fleksibiliteten til Web services og åpen teknologi.

SAP NetWeaver er selve grunnpilaren i alle SAPs løsninger, og tilbyr dermed komplett organisasjonsintegrasjon på flere nivåer. Nøkkelkonseptene for SAP NetWeaver kan i grove trekk deles inn i følgende kategorier; menneskelig integrasjon, informasjonsintegrasjon, prosessintegrasjon, applikasjonsplattform, håndtering av livssyklus, samt et rammeverk for sammensatte applikasjoner.

Applikasjonsplattformen støtter språkene Java 2 Platform, Enterprise Edition (J2EE) og Advanced Business Application Programming (ABAP) i et enkelt kjøringsmiljø, og SAP NetWeaver tilbyr uavhengighet fra eksisterende databaser og operativsystem, full støtte til Web services og virksomhetsapplikasjoner og et utviklingsmiljø basert på åpne standarder.

For å håndtere livssyklusen til et system tilbyr SAP NetWeaver teknologi for å håndtere alle stegene i programvarens livssyklus, fra design, utrulling, implementasjon, versjonshåndtering og testing gjennom operasjoner som administrasjon og endringshåndtering. Rammeverket for sammensatte applikasjoner tilbyr på sin side et utviklingsmiljø for å bygge SAP xApps sammensatte applikasjoner. SAP NetWeaver støtter dette ved å tilby verktøy, metodologier, regler og mønstre for å støtte utviklingen av slike sammensatte applikasjoner.

A.3 CORBA

CORBA er en annen teknologi som av mange er brukt for å realisere tjenesteorienterte arkitekturer. Fordelene med å bruke CORBA for å implementere en tjenesteorientert arkitektur er blant annet at CORBA er en åpen standard, og dermed lett tilgjengelig (Newcomer 2004). I tillegg støtter CORBA asynkrone meldinger, og publish/subscribe-kommunikasjon (tilbud/abonnement). CORBA har integrerte sikkerhetsmekanismer, navngiving av tjenester, transaksjonshåndtering og pålitelig meldingsutveksling, noe som helt klart er en fordel for en tjenesteorientert arkitektur. CORBA har også støtte for flere ulike programmeringsspråk, og tilbyr blant annet CORBA IDL, et eget språk for å definere tjenester. En annen stor fordel med CORBA er at CORBA-objekter kan uttrykkes som Web services fordi Object Management Group (OMG) har definert en CORBA IDL til WSDL-mapping.

Det er naturlig nok ikke bare fordeler knyttet til å bruke CORBA i en tjenesteorientert arkitektur. En ulempe er for eksempel at CORBA regnes for å være relativt tung og kompleks, og dermed vanskelig å lære dersom man ikke har brukt denne teknologien tidligere (Newcomer 2004). I tillegg, og en kanskje enda støtte ulempe, er at CORBA krever at både bruker og tilbyder av en tjeneste må benytte CORBA, i tillegg til at CORBA heller ikke har noen umiddelbar støtte for XML, eller løst koblet asynkron utveksling av dokumenter over Internett.

Man må imidlertid være oppmerksom på at selv om CORBA, i likhet med WebSphere MQ og SAP NetWeaver, kan benyttes for å utvikle tjenesteorienterte arkitekturer, er det ikke dermed slik at alle systemer utviklet med en av disse teknologiene er tjenesteorienterte arkitekturer.

A.4 Web services

Web services er løst koblede nettverksgrensesnitt som beskrives ved hjelp av et XML basert dokument. Dette grensesnittet tillater tjenesten å bli brukt uavhengig av plattform og programmeringsspråk (Hogg, Chillcott, Nolan & Srinivasan 2004). Web services kan sees som en måte å designe sømløse og fleksible interaksjoner over applikasjoner på, både innen og på tvers av organisasjoner. Selve termen "Web services" viser til en gruppe teknologier som gjør forretningsprosesser eller forretningsinformasjon tilgjengelig over Internett (Moitra & Ganesh 2004). Web services inneholder funksjonalitet og informasjon, og grensesnittene er programmerbare grensesnitt, i motsetning til grensesnitt for vanlige web-applikasjoner (Chung 2005).

Web services er XML-baserte teknologier for å sende og motta meldinger, beskrive og oppdage tjenester (Newcomer 2004). I tillegg er Web services basert på åpne standarder for å distribuere grensesnitt og dokumenter enkelt via meldinger. Web services tilbyr utvidbarhet for ulike kvalitetsattributter som sikkerhet, pålitelighet og transaksjoner, og støtte for sammensatte applikasjoner.

De største fordelene med å implementere en tjenesteorientert arkitek-

tur med Web services er at Web services etter hvert er utbredt, enkelt å bruke og å lære, samt at teknologien i seg selv er plattformnøytral. I tillegg impliserer Web services sterke bånd til forretningsprosesser, samt at koblingen mellom grensesnittene og den tekniske implementasjonen er så løs at samarbeid med ulike partnere med ulike programvareløsninger blir langt enklere enn med for eksempel klient/tjenerløsninger, hvor man er avhengig av integrerte forretningssystemer eller spesialløsninger for hver enkelt samarbeidspartner (Newcomer 2004). Fordi Web services oppfordrer til teknologisk frihet i forhold til applikasjonsutvikling, blir ikke Web services i seg selv synlig for den enkelte bruker. Mange mener dette er en av grunnene til at en del ledere ennå ikke har forstått hvilket vendepunkt og hvilke muligheter som ligger i Web services både for næringslivet og for offentlig sektor. Alonso & Casati (2005) hevder at Web services og tjenesteorienterte arkitekturer er et naturlig valg for å implementere distribuerte systemer og ved applikasjonsintegrasjon på tvers av virksomhetsgrenser.

Se forøvrig tabell A.1 på neste side for en oppsummering av teknologiene nevnt over med utvalgte egenskaper.

Egenskap for plattformen	**WebSphere MQ**	**CORBA**	**SAP NetWeaver**	**Web Services**
Åpne standarder	Nei	X	X	X
Støtte for flere programmerings-språk	X	X	X	X
Løst koplet	X	delvis	X	X
XML-basert	mulig	mulig	mulig	X
Teknologinøytrale definisjoner av tjenestekontrakter	diverse	X	mulig	X
Meldings- og parameter-definisjoner	X	X	mulig	X
Meldings- og parameter-valideringer	må spesial-designes	X	mulig	X
Meldings- og parameter-parsing	må spesial-designes	X	mulig	X
Autentisering	X	X	X	X
Autorisasjon	ingen	X	X	X
Personvern	ingen	X	X	X
Integritet	ingen	X	X	X
En-veis synkron kommunikasjon	X	X	X	X
Spørring/svar	svak støtte	X	X	X
En-veis asynkron kommunikasjon	X	svak støtte	X	X
Offentliggjøre/ Abonnere	svak støtte	X	X	X
Konvertering av data mellom plattformer	svak støtte	X	X	behøver ikke

Tabell A.1: Ulike teknologier og egenskaper for realisering av en tjenesteorientert arkitektur

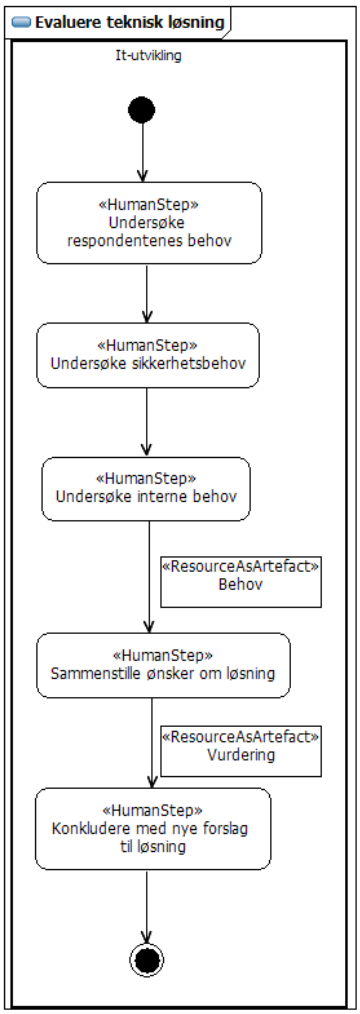
Tillegg B

Fullstendig systemintegrasjon av Kostra og Idun

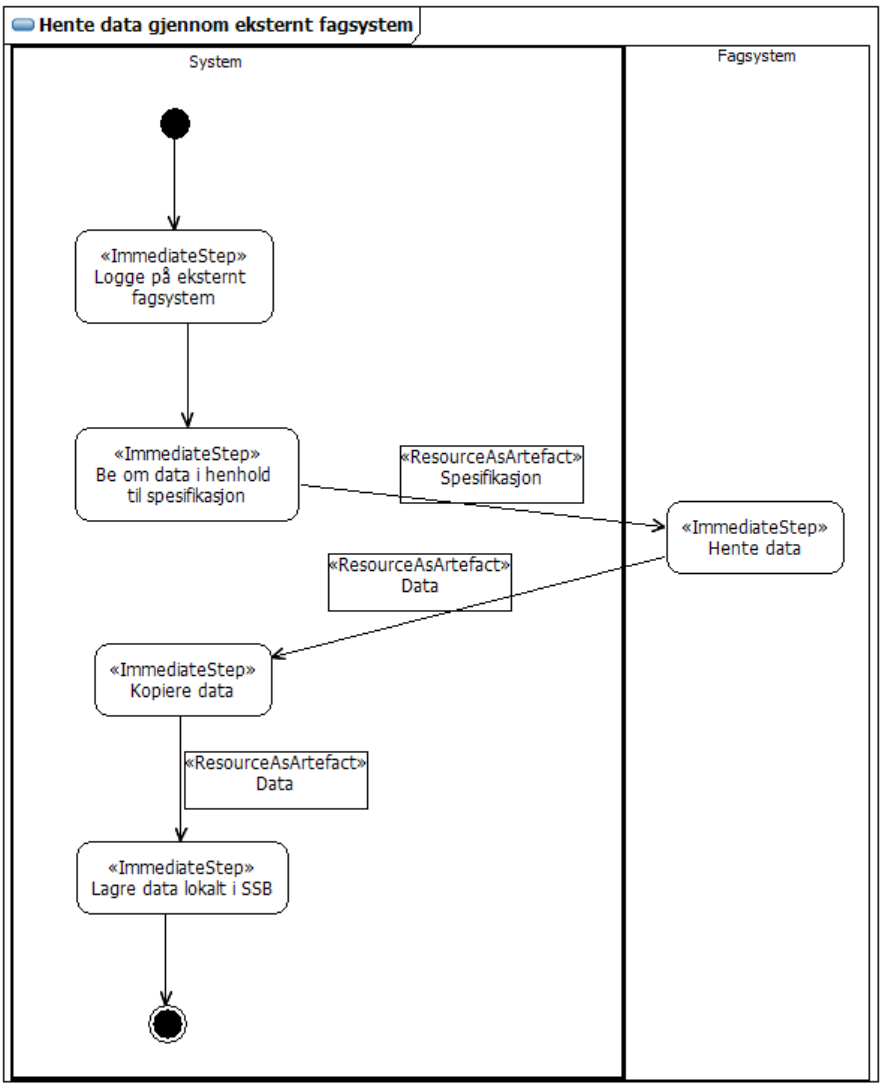
Her følger en detaljering av de øvrige modellene som inngår i systemintegrasjonen av Kostra og Idun, beskrevet i kapittel 3 fra side 35

B.1 Virksomhetsmodeller

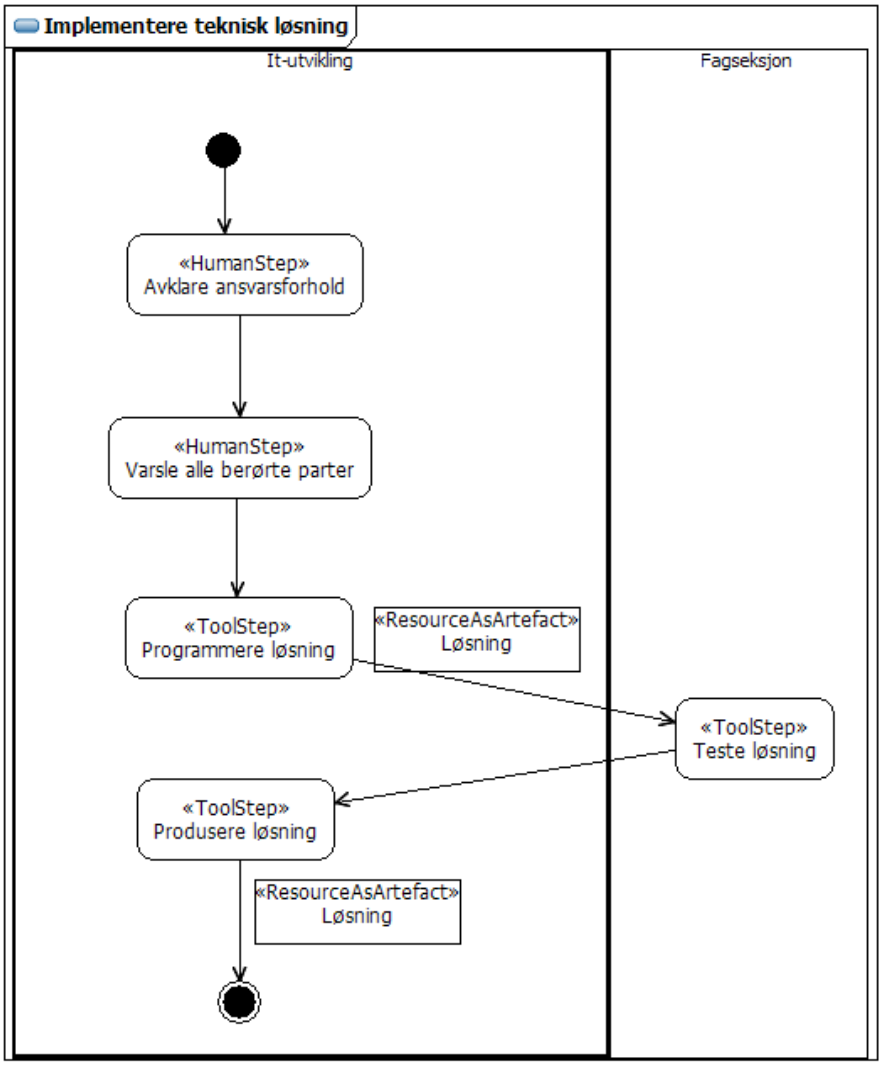
Her følger de øvrige prosessmodellene utledet fra prosess- og målmodellen, se figur 3.5 på side 53.



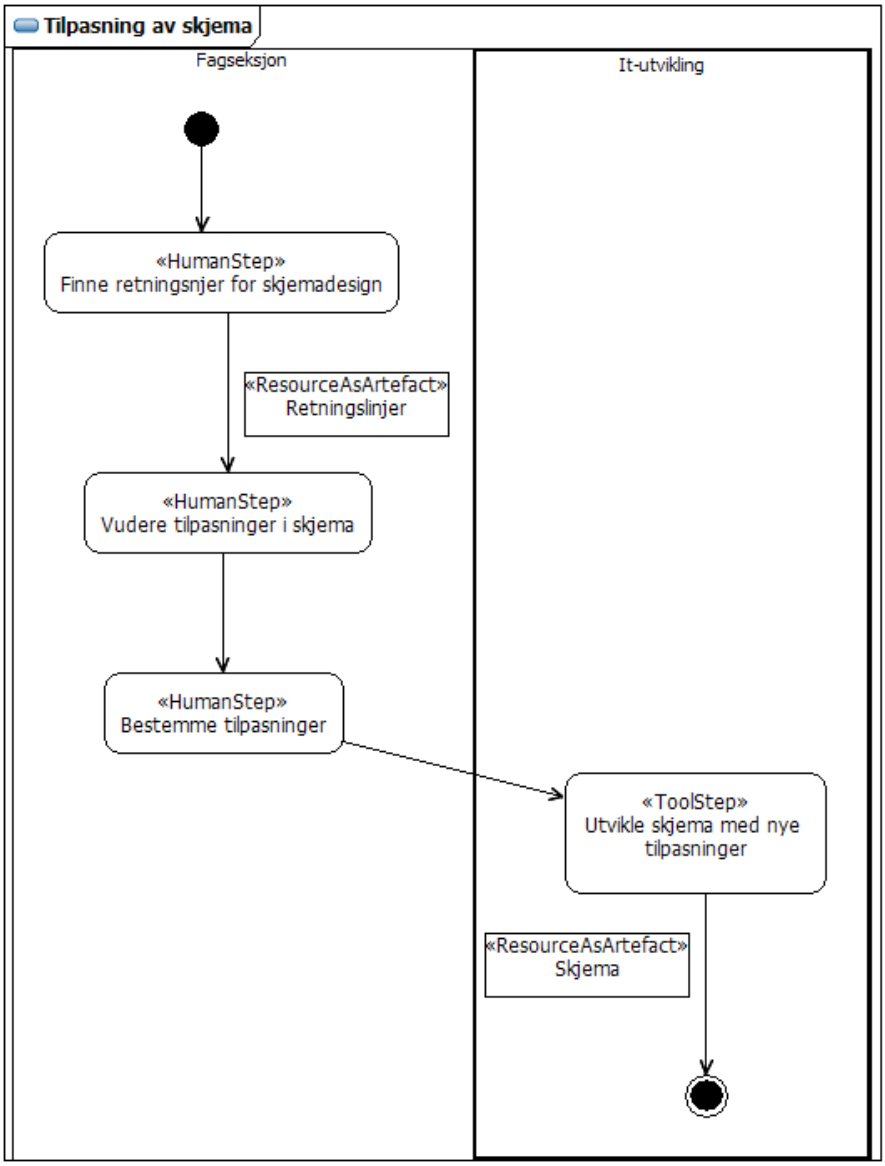
Figur B.1: Prosessen "Evaluere teknisk løsning"



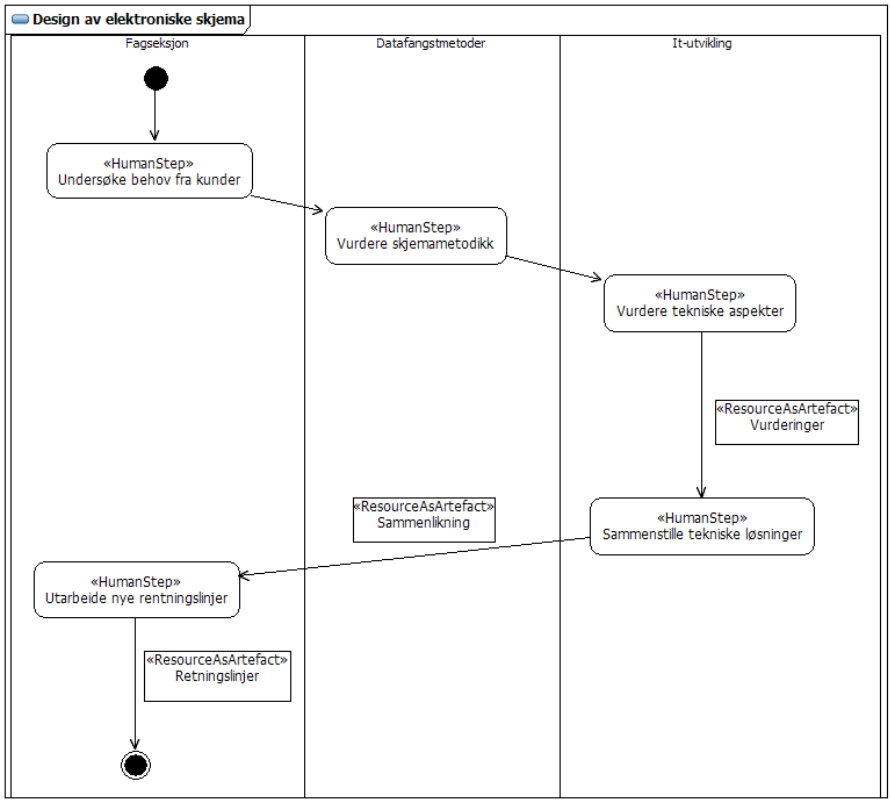
Figur B.2: Prosessen "Hente data fra eksternt fagsystem"



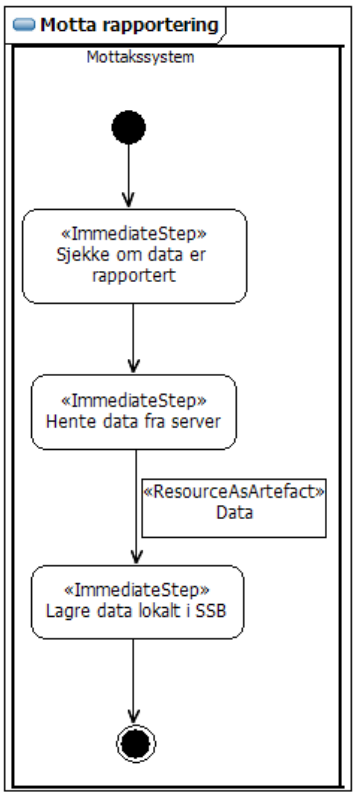
Figur B.3: Prosessen "Implementere Teknisk Løsning"



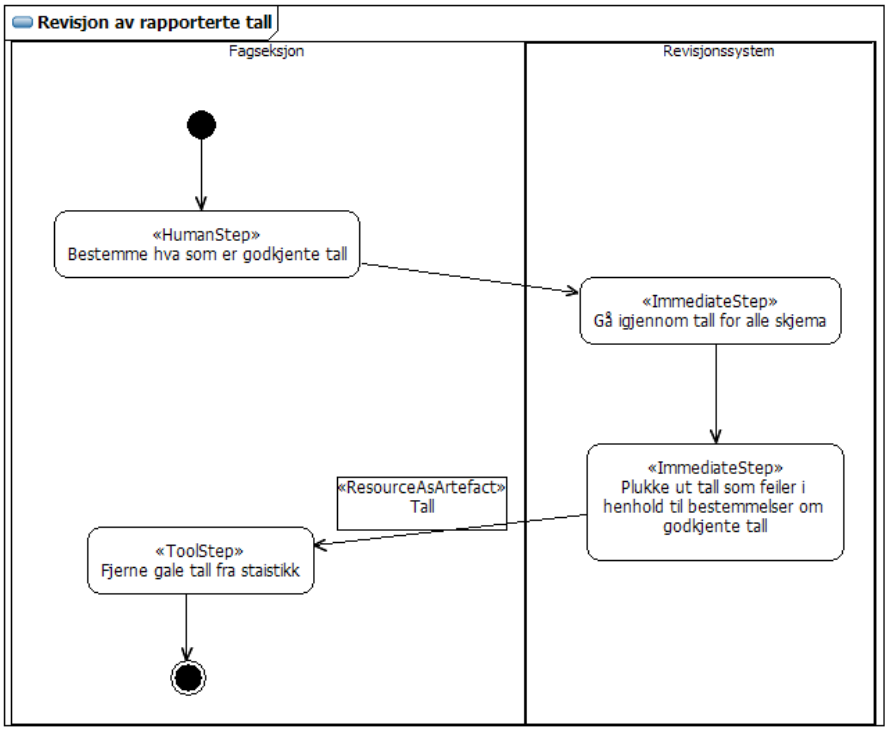
Figur B.4: Prosessen "Tilpasning av skjema"



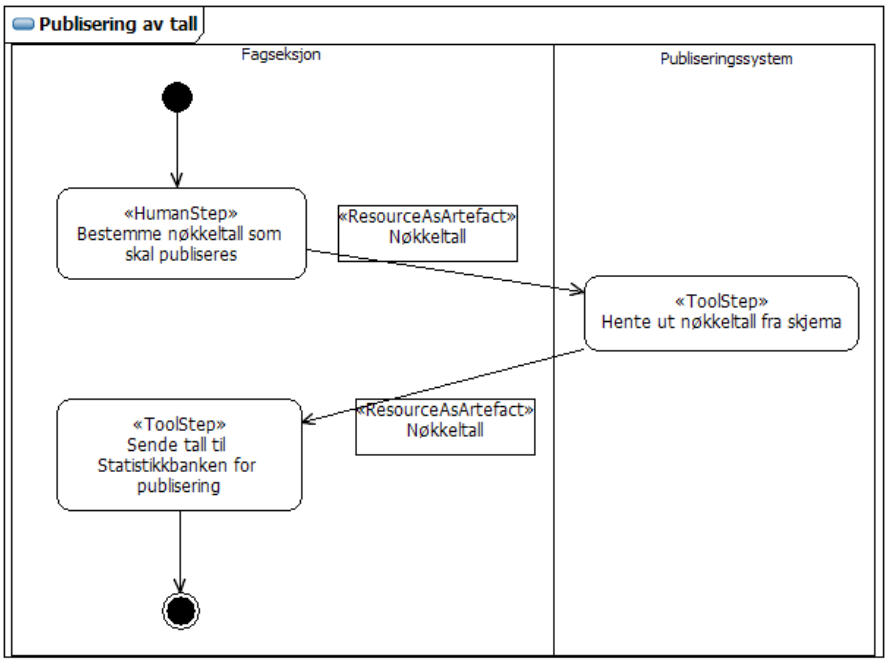
Figur B.5: Prosessen “Design av elektroniske skjema”



Figur B.6: Prosessen "Motta rapportering"



Figur B.7: Prosessen "Revisjon av rapporterte tall"



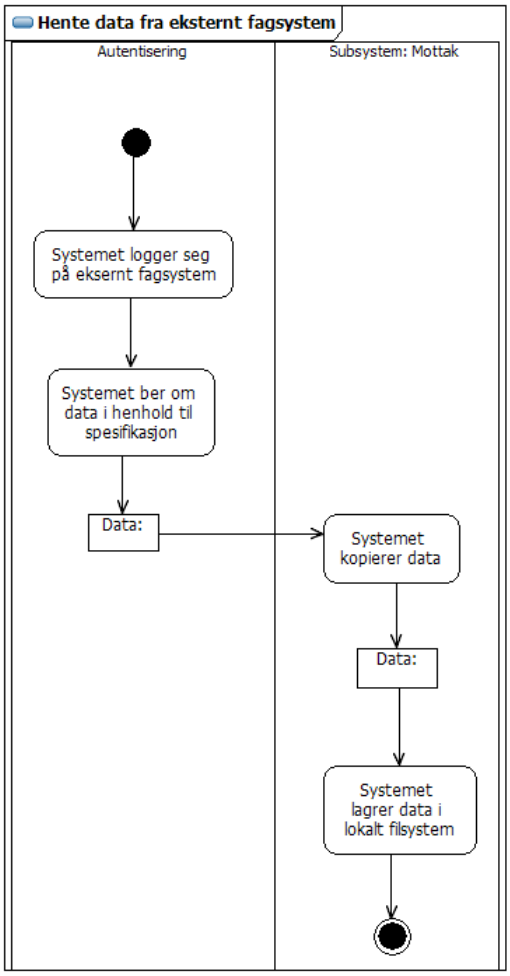
Figur B.8: Prosessen "Publisering av tall"

B.2 Kravmodeller

Kravmodellene er utledet fra use case modellen (se figur 3.7 på side 55). Hvert av use casene er detaljert i form av en use case beskrivelse og en aktivitetsmodell for å beskrive meldingsflyt og ansvarsforhold i virksomheten.

Use Case:	**Hente data fra eksternt fagsystem**
Mål:	Hente data fra fagsystem
Pre-betingelser:	Fagsystem er tilgjengelig
Post-betingelser:	Data er lastet over til et lokalt filsystem i SSB
Normal hendelsesflyt:	
1.	Systemet logger seg inn på et eksternt fagsystem
2.	Systemet ber om data i henhold til en spesifikasjon
3.	Fagsystem henter data
4.	Systemet kopierer data
5.	Systemet lagrer data i et lokalt filsystem
Variasjoner:	
1a.	Feil under innlogging
1a1.	Systemet prøver å logge på en gang til
1a2.	Use Case fortsetter fra punkt 2
3a.	Fagsystem kan ikke finne data i henhold til spesifikasjon
3a1.	Use Case avsluttes
5a.	Systemet kan ikke lagre data lokalt
5a1.	Use Case avsluttes

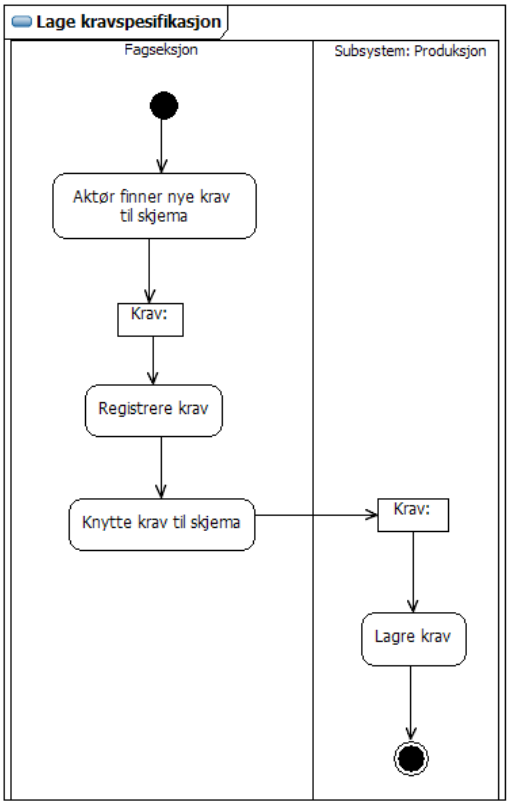
Tabell B.1: Beskrivelse av use caset "Hente data fra eksternt fagsystem"



Figur B.9: Aktivitetsmodell for use caset "Hente data fra eksternt fagsystem"

Use Case:	**Lage kravspesifikasjon**
Mål:	Utarbeide krav til skjema
Pre-betingelser:	Fagseksjonen opplever et behov for et nytt spørreskjema eller endringer i et eksisterende spørreskjema
Post-betingelser:	Ferdig kravspesifikasjon foreligger
Normal hendelsesflyt:	
1.	Fagseksjon finner nye krav til skjema
2.	Fagseksjonen dokumenterer krav til skjema
Variasjoner:	ingen

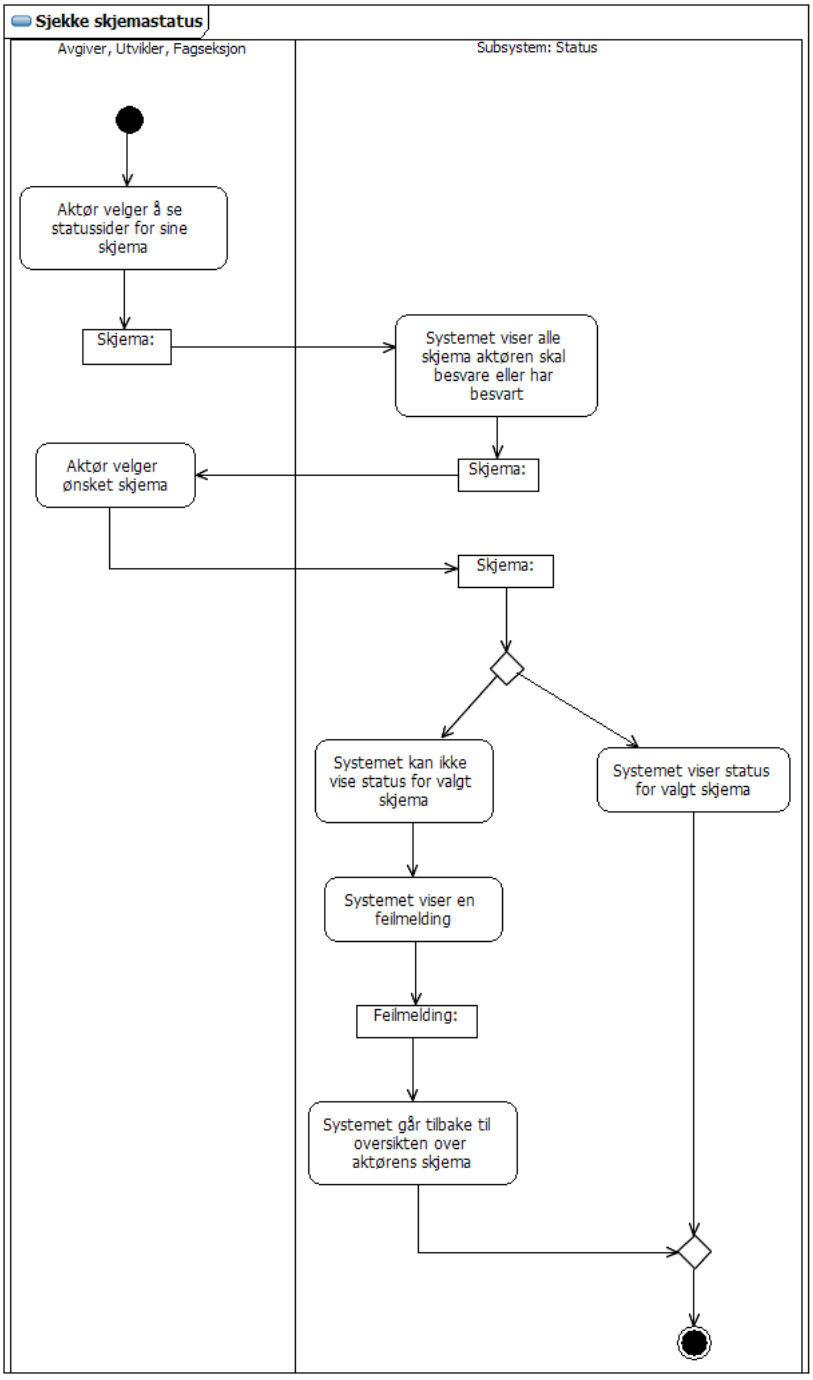
Tabell B.2: Beskrivelse av use caset "Lage kravspesifikasjon"



Figur B.10: Aktivitetsmodell for use caset "Lage kravspesifikasjon"

Use Case:	**Sjekke skjemastatus**
Aktører:	Avgiver, Utvikler, Fagseksjon
Mål:	Finne status for et gitt skjema
Pre-betingelser:	Systemet viser aktørens tilgjenglige skjema
Post-betingelser:	Status for et gitt skjema er sjekket
Normal hendelsesflyt:	
1.	Aktør velger å se statussider for sine skjema
2.	System viser alle skjema aktøren skal besvare eller har besvart
3.	Aktør velger ønsket skjema
4.	Systemet viser status for valgt skjema
Variasjoner:	
4a.	Systemet kan ikke vise valgt skjema
4a1.	Systemet viser en feilmelding
4a2.	Systemet går tilbake til oversikt over aktørens skjema

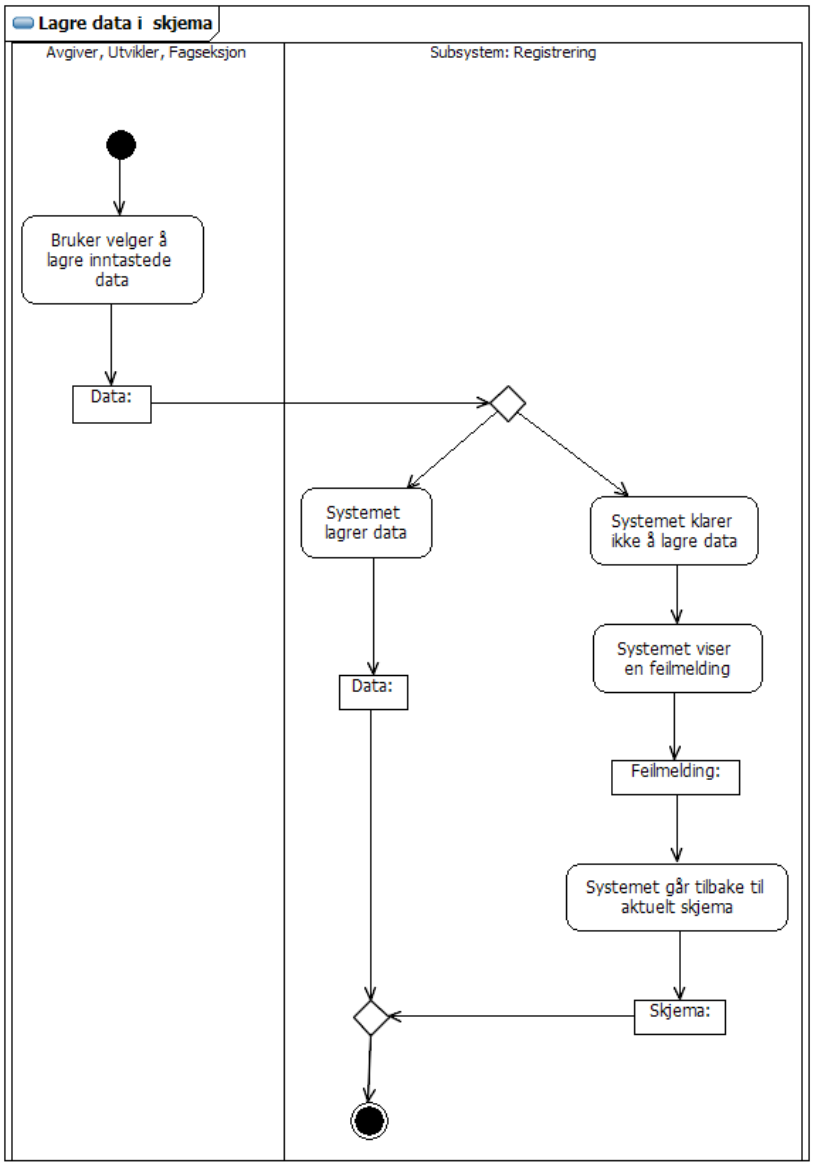
Tabell B.3: Beskrivelse av use caset "Sjekke skjemastatus"



Figur B.11: Aktivitetsmodell for use caset “Sjekke skjemastatus”

Use Case:	**Lagre data i skjema**
Aktører:	Avgiver, Fagseksjon, Utvikler
Mål:	Data som aktør har registrert i skjemaet er lagret i systemet
Pre-betingelser:	Aktør har tastet inn data i systemet
Post-betingelser:	Data er lagret
Normal hendelsesflyt:	
1.	Aktør velger å lagre inntastede data
2.	Systemet lagrer data i tilknytning til bruker og skjema
Variasjoner:	
2a.	Systemet klarer ikke å lagre data
2a1.	Systemet viser en feilmelding

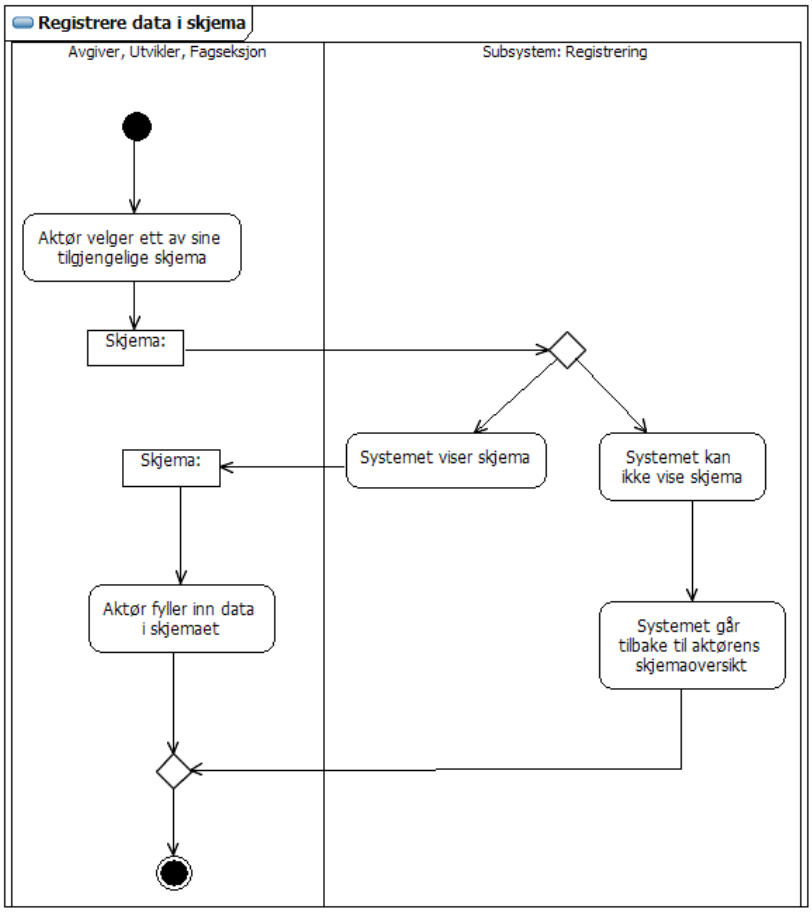
Tabell B.4: Beskrivelse av use caset "Lagre data i skjema"



Figur B.12: Aktivitetsmodell for use caset “Lagre data i skjema”

Use Case:	**Registrere data i skjema**
Aktører:	Avgiver, Utvikler, fagseksjon
Mål:	Fylle inn data i spørreskjema
Pre-betingelser:	Aktør ser sine tilgjengelige skjema
Post-betingelser:	Skjema inneholder data fra aktør
Normal hendelsesflyt:	
1.	Aktør velger ett av sine tilgjengelige skjema
2.	Systemet viser valgt skjema
3.	Aktør fyller inn data i skjemaet
Variasjoner:	
2a.	Systemet kan ikke vise valgt skjema
2a1.	Systemet viser en feilmelding
2a2.	Systemet går tilbake til aktørens skjema-oversikt

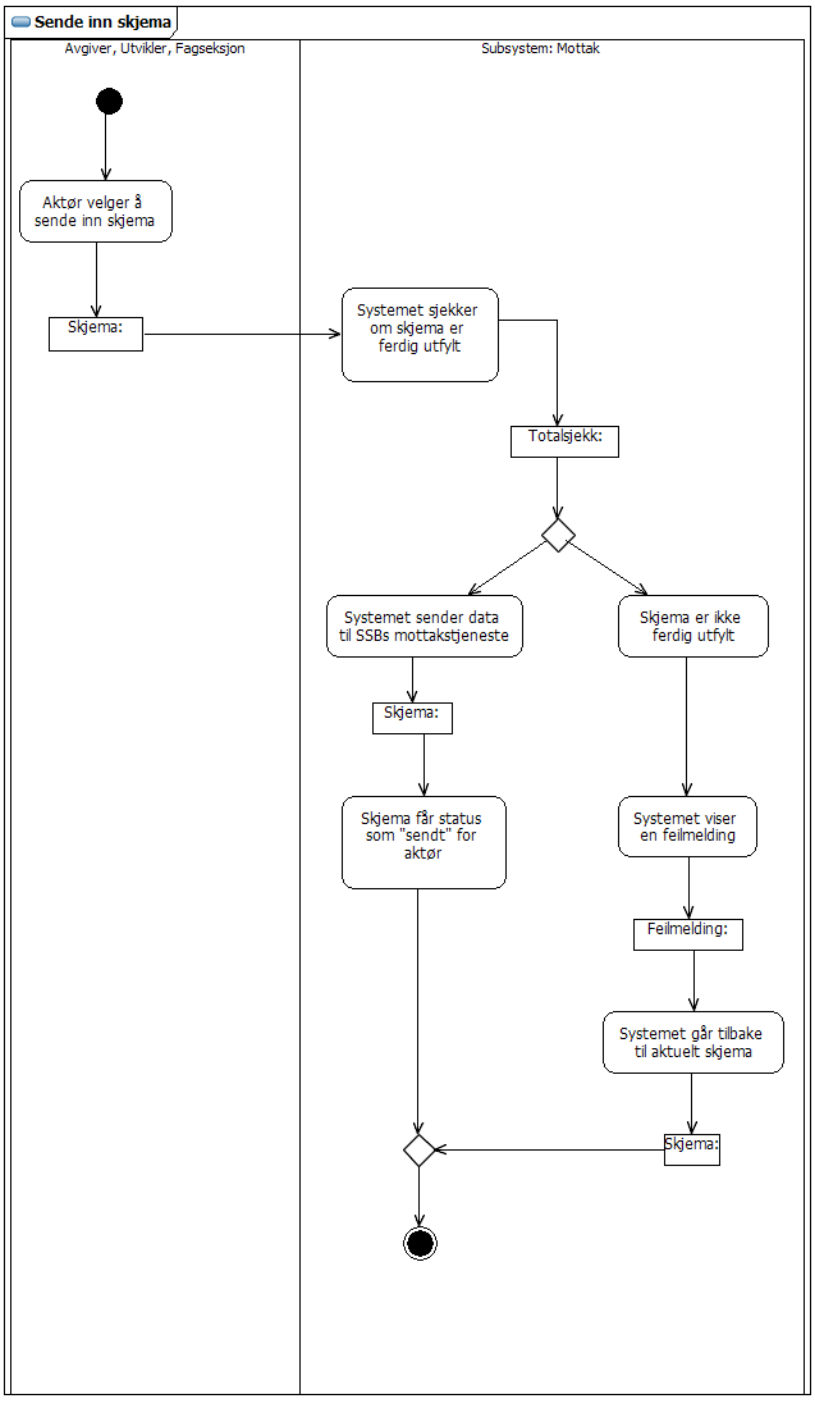
Tabell B.5: Beskrivelse av use caset "Registrere data i skjema"



Figur B.13: Aktivitetsmodell for use caset "Registrere data i skjema"

Use Case:	**Sende inn skjema**
Aktører:	Avgiver, Fagseksjon, Utvikler
Mål:	Registrere data i SSBs mottakssystem
Pre-betingelser:	Aktør har fylt ut skjema
Post-betingelser:	Skjema har status som "innsendt"
Normal hendelsesflyt:	
1.	Aktør velger å sense inn skjema
2.	Systemet sjekker om skjema er ferdig utfylt
3.	Systemet sender data til SSBs mottaks-tjeneste
4.	Skjema får status som "sendt" for aktør
Variasjoner:	
2a.	Skjema er ikke ferdig utfylt
2a1.	Systemet viser en feilmelding
2a2.	Systemet går tilbake til aktuelt skjema

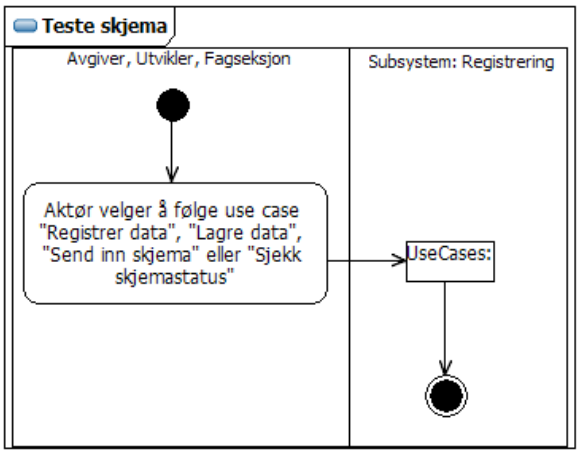
Tabell B.6: Beskrivelse av use caset "Sende inn skjema"



Figur B.14: Aktivitetsmodell for use caset “Sende inn skjema”

Use Case:	**Teste skjema**
Aktører:	Utvikler, Fagseksjon, Avgiver
Mål:	Finne feil i skjema
Pre-betingelser:	Skjema er publisert for testing
Post-betingelser:	Skjema er testet
Normal hendelsesflyt:	
1.	Aktør velger use case “Registrere data i skjema”, “Lagre data i skjema”, “Sende inn skjema”, “Sjekke skjemastatus”, “Vise tilgjengelige skjema” eller “Hente tidligere lagrede data”
Variasjoner:	ingen

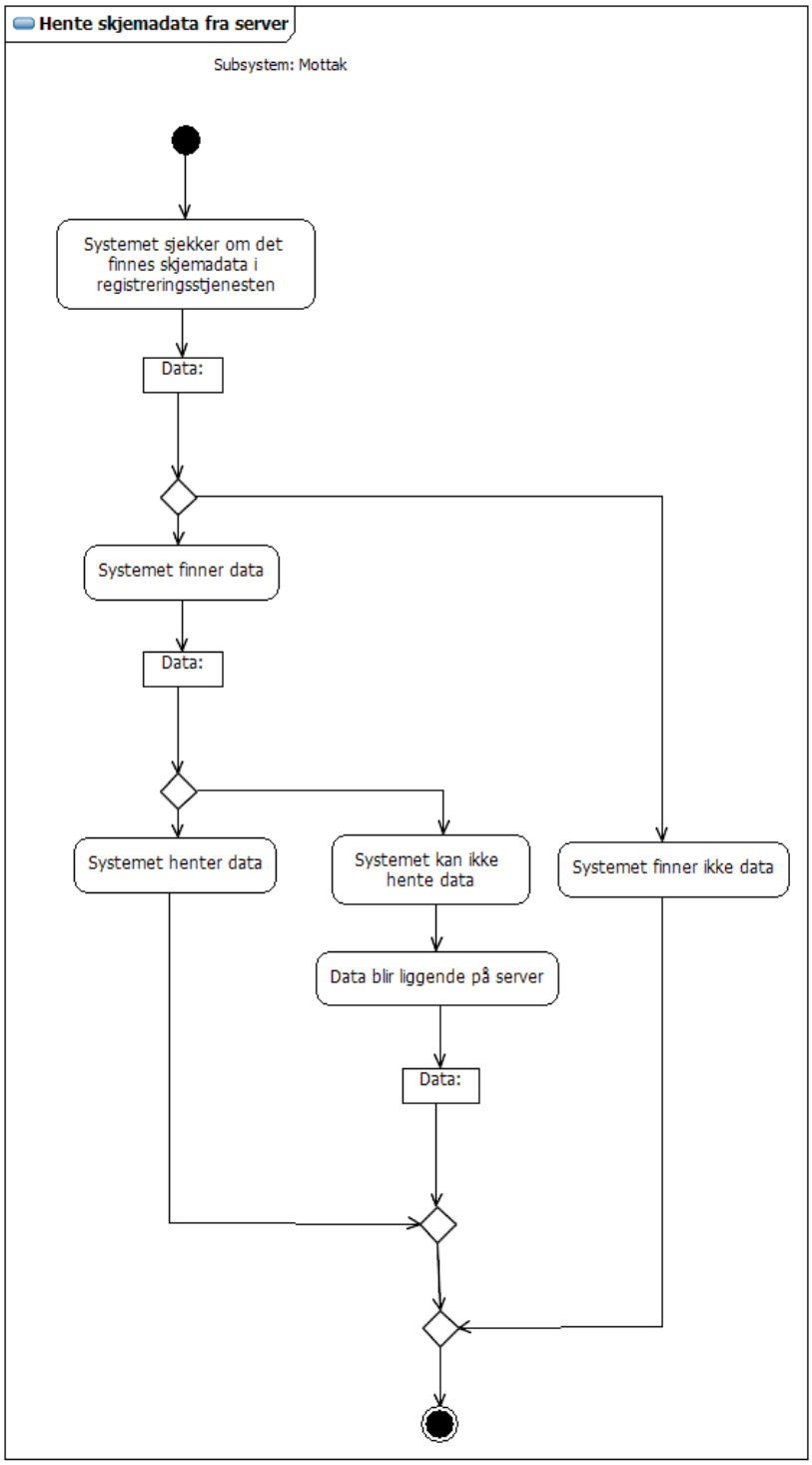
Tabell B.7: Beskrivelse av use caset “Teste skjema”



Figur B.15: Aktivitetsmodell for use caset “Teste skjema”

Use Case:	**Hente skjemadata fra server**
Aktører:	System
Mål:	Hente data fra registreringstjenesten
Pre-betingelser:	Systemet er oppe
Post-betingelser:	Data er hentet
Normal hendelsesflyt:	
1.	Systemet sjekker om det finnes skjemadata i registreringstjenesten
2.	Systemet finner data
3.	Systemet henter data
Variasjoner:	
2a.	Systemet finner ikke data, use case avsluttes
3a.	Systemet kan ikke hente data
3a1.	Data blir liggende på server, use case avsluttes

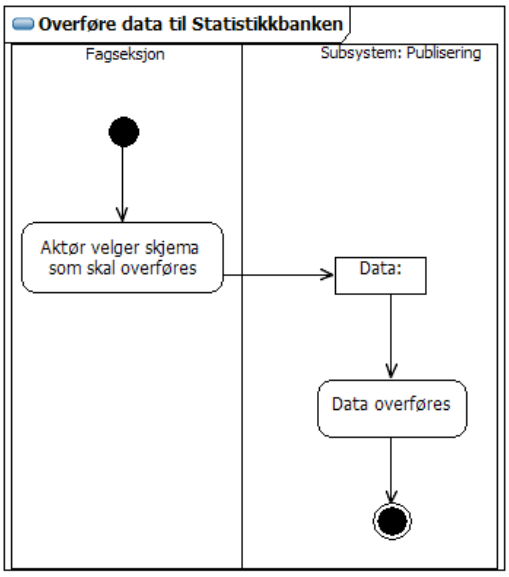
Tabell B.8: Beskrivelse av use caset "Hente skjemadata fra server"



Figur B.16: Aktivitetsmodell for use caset "Hente skjemadata fra server"

Use Case:	Overføre data til Statistikkbanken
Aktører:	Fagseksjon
Mål:	Overføre data til Statistikkbanken
Pre-betingelser:	Skjema er lagret på forhåndsbestemt tjener
Post-betingelser:	Data er lagret i Statistikkbanken
Normal hendelsesflyt:	
1.	Aktør velger skjema som skal overføres
2.	Data overføres
Variasjoner:	ingen

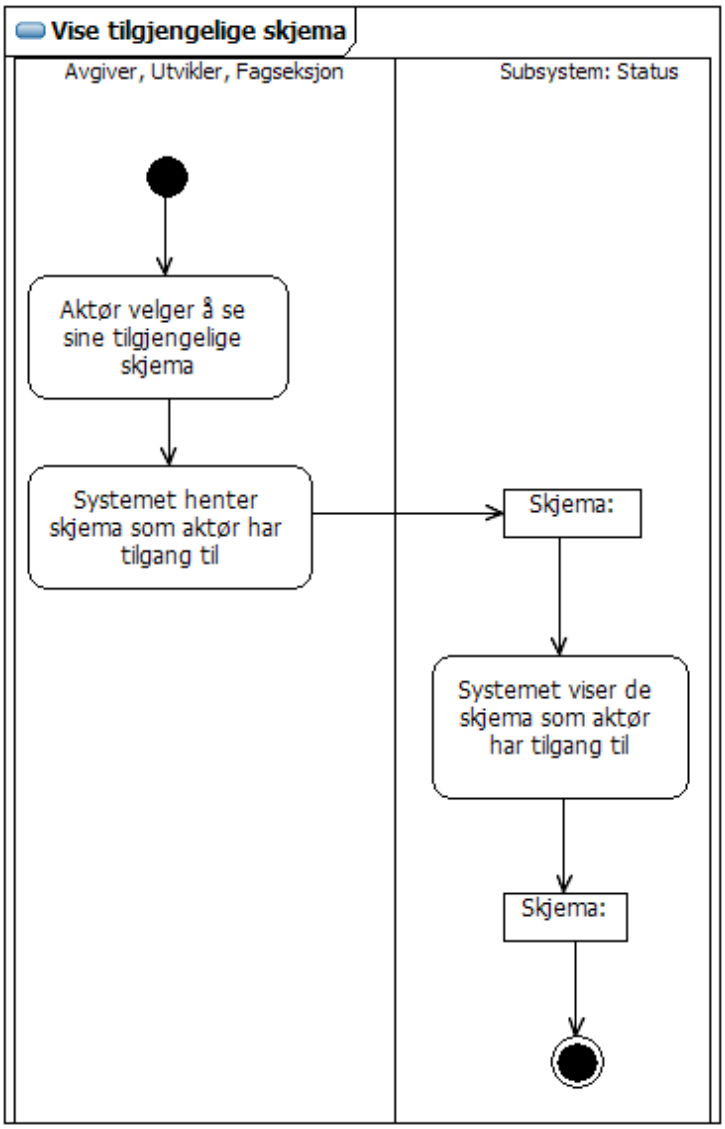
Tabell B.9: Beskrivelse av use caset "Overføre data til Statistikkbanken"



Figur B.17: Aktivitetsmodell for use caset "Overføre data til Statistikkbanken"

Use Case:	**Vise tilgjengelige skjema**
Aktører:	Avgiver, Utvikler, Fagseksjon
Mål:	Vise de skjema som er tilgjengelige fra avgiver
Pre-betingelser:	Aktør må være logget på systemet
Post-betingelser:	Aktør kan se alle sine tilgjengelige skjema
Normal hendelsesflyt:	
1.	Aktør velger å vise sine tilgjengelige skjema
2.	Systemet henter de skjema som aktør har tilgang til
3.	Systemet viser de skjema som aktør har tilgang til
Variasjoner:	ingen

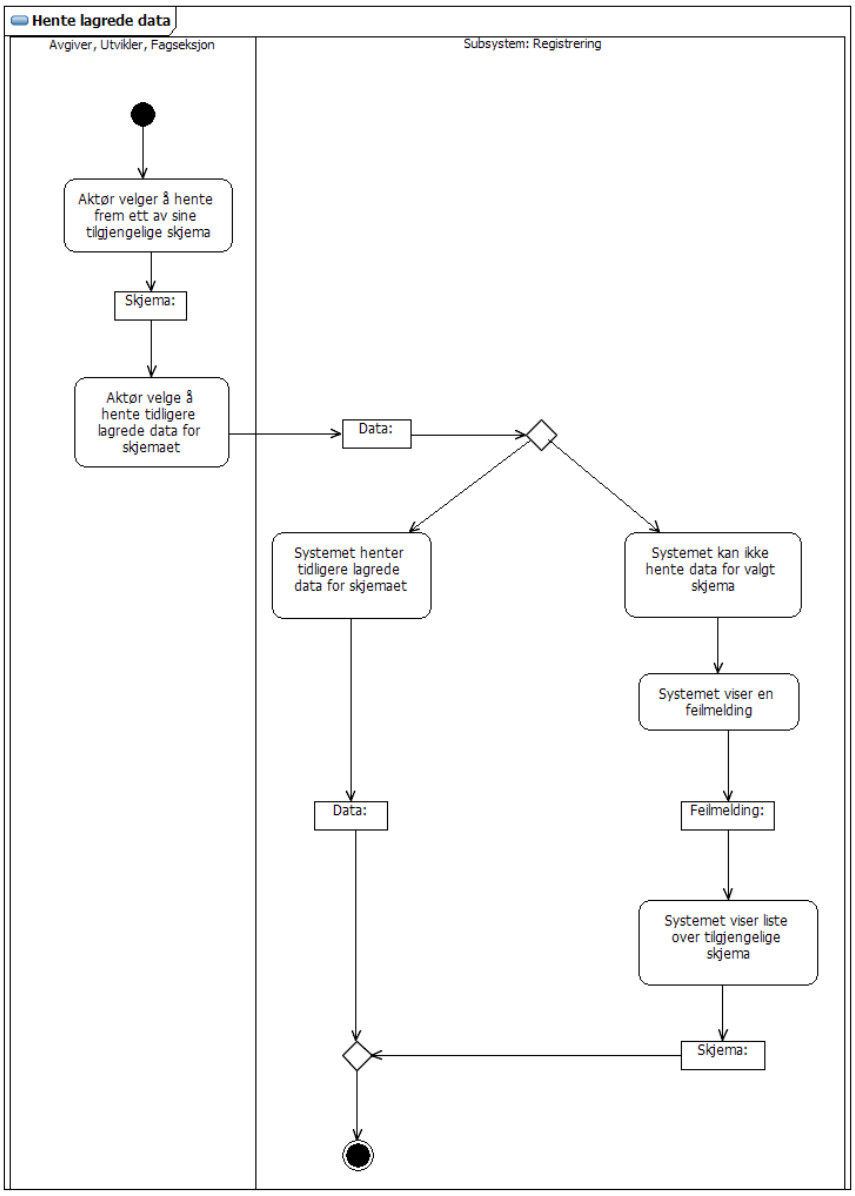
Tabell B.10: Beskrivelse av use caset "Vise tilgjengelige skjema"



Figur B.18: Aktivitetsmodell for use caset "Vise tilgjengelige skjema"

Use Case:	**Hente tidligere lagrede data i skjema**
Aktører:	Avgiver, Utvikler, Fagseksjon
Mål:	Å hente data som er registrert i skjema på et tidligere tidspunkt
Pre-betingelser:	Aktør ser sine tilgjengelige skjema
Post-betingelser:	Tidligere lagrede data er hentet frem i valgt skjema
Normal hendelsesflyt:	
1.	Aktør velger ett av sine tilgjengelige skjema
2.	Aktør velger å hente lagrede data for skjemaet
3.	Systemet henter tidligere lagrede data for skjemaet
Variasjoner:	
3a.	Systemet kan ikke hente data
3a1.	Systemet viser en feilmelding
3a2.	Systemet viser liste over tilgjengelige skjema

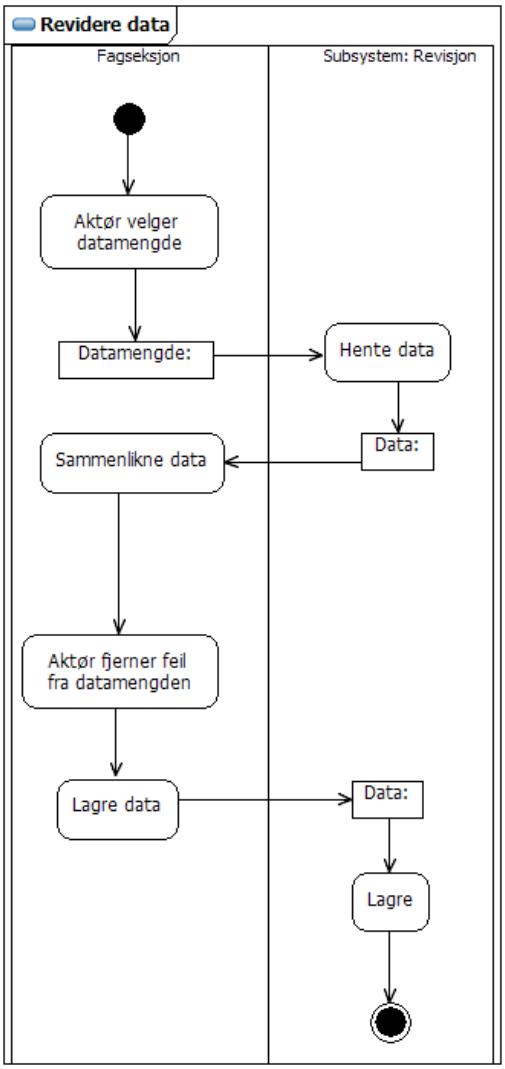
Tabell B.11: Beskrivelse av use caset "Hente tidligere lagrede data for skjema"



Figur B.19: Aktivitetsmodell for use caset “Hente tidligere lagrede data for skjema”

Use Case:	**Revidere data**
Aktører:	Fagseksjon
Mål:	Fjerne feil i data
Pre-betingelser:	Data fra skjema ligger lagret lokalt hos SSB
Post-betingelser:	Datamengden er fri for feil
Normal hendelsesflyt:	
1.	Aktør velger datamengde
2.	Aktør fjerner feil fra datamengden
Variasjoner:	ingen

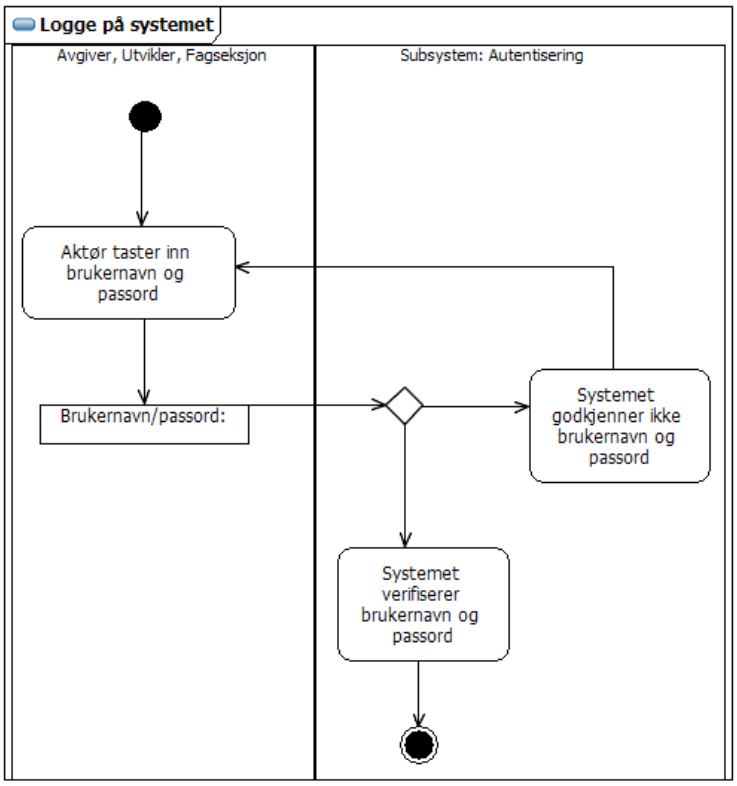
Tabell B.12: Beskrivelse av use caset "Revidere data"



Figur B.20: Aktivitetsmodell for use caset "Revidere data"

Use Case:	**Logge på systemet**
Aktører:	Avgiver, Utvikler, Fagseksjon
Mål:	Logge på systemet
Pre-betingelser:	Aktør ønsker å logge på systemet
Post-betingelser:	Aktør er logget på systemet
Normal hendelsesflyt:	
1.	Aktør taster inn brukernavn og passord
2.	Systemet verifiserer brukernavn og passord
Variasjoner:	
2a.	Systemet godkjenner ikke brukernavn og passord
2a1.	Use caset starter fra punkt 1

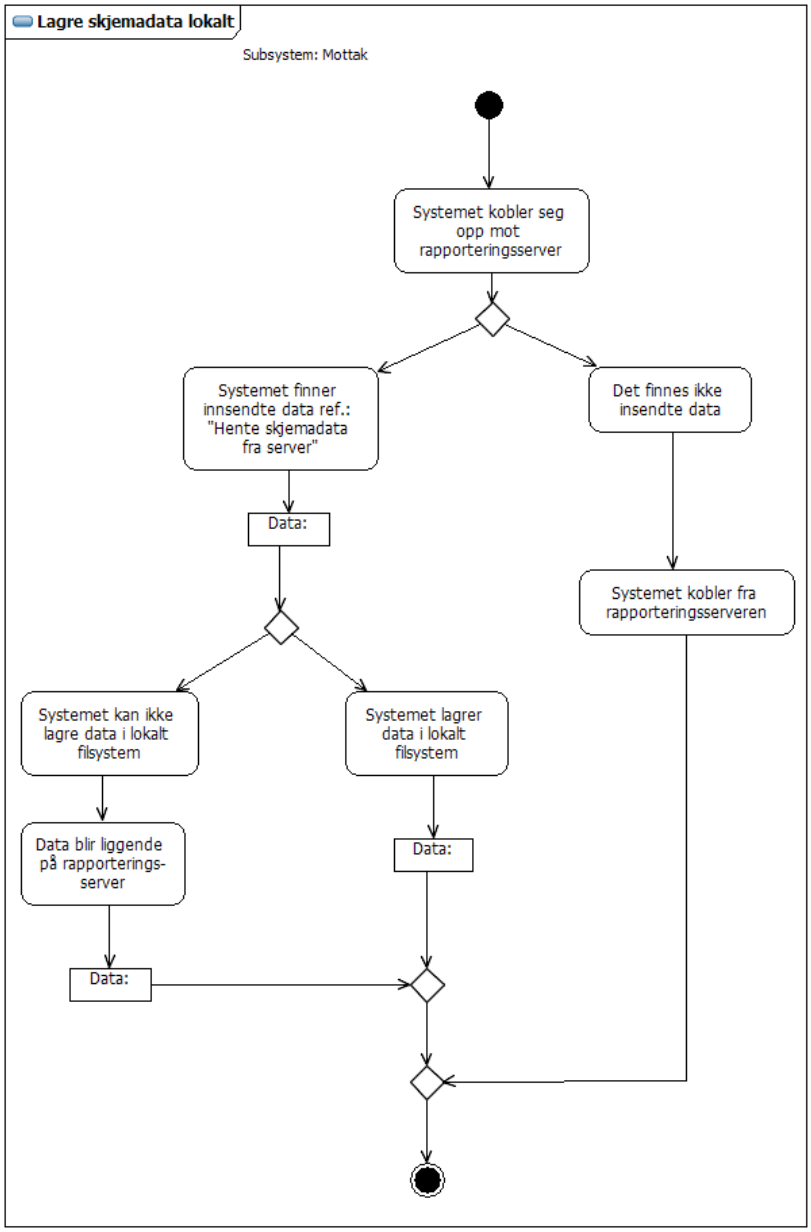
Tabell B.13: Beskrivelse av use caset "Logge på systemet"



Figur B.21: Aktivitetsmodell for use caset "Logge på systemet"

Use Case:	**Lagre skjemadata lokalt**
Aktører:	Systemet
Mål:	Lagre data fra server i lokalt filsystem i SSB
Pre-betingelser:	Systemet er oppe
Post-betingelser:	Systemet er koblet fra rapporterings-server
Normal hendelsesflyt:	
1.	Systemet kobler seg opp mot rapporteringsserver
2.	Systemet finner innsendte data se tabell B.8 på side 135
3.	Systemet lagrer data i lokalt filsystem
Variasjoner:	
2a.	Det finnes ikke innsendte data
2a1.	Systemet kobler fra rapporteringsserveren
3a.	Systemet kan ikke lagre data i lokalt filsystem
3a1.	data blir liggende på rapporteringsserver

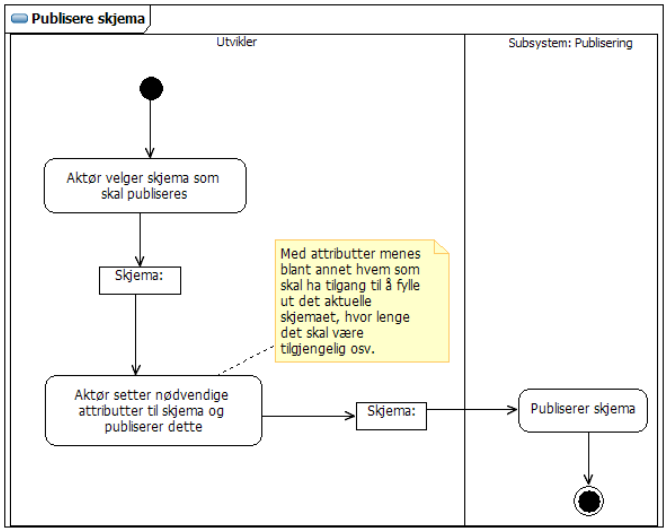
Tabell B.14: Beskrivelse av use caset “Lagre skjemadata lokalt”



Figur B.22: Aktivitetsmodell for use caset “Lagre skjemadata lokalt”

Use Case:	**Publisere skjema**
Aktører:	Utvikler
Mål:	Gjøre skjema tilgjengelige for test eller innrapportering
Pre-betingelser:	Skjema er produsert
Post-betingelser:	Skjema er publisert
Normal hendelsesflyt:	
1.	Aktør velger skjema som skal publiseres
2.	Aktør setter nødvendige attributter til skjema og publiserer dette
Variasjoner:	ingen

Tabell B.15: Beskrivelse av use caset "Publisere skjema"



Figur B.23: Aktivitetsmodell for use caset "Publisere skjema"